Xpert.press

Gregor Sandhaus · Björn Berg
Philip Knott

Hybride Softwareentwicklung

Das Beste aus klassischen und agilen Methoden in einem Modell vereint

Springer Vieweg

Xpert.press

Weitere Bände in dieser Reihe
http://www.springer.com/series/4393

Die Reihe **Xpert.press** vermittelt Professionals in den Bereichen Softwareentwicklung, Internettechnologie und IT-Management aktuell und kompetent relevantes Fachwissen über Technologien und Produkte zur Entwicklung und Anwendung moderner Informationstechnologien.

Gregor Sandhaus • Björn Berg • Philip Knott

Hybride Softwareentwicklung

Das Beste aus klassischen und agilen Methoden in einem Modell vereint

Springer Vieweg

Prof. Dr. Gregor Sandhaus
Wirtschaftsinformatik
Hochschule für Oekonomie und Management
Essen
Deutschland

Philip Knott
Informationsverarbeitung
und Telekommunikation
DEVK Deutsche Eisenbahn Versicherung
Köln
Deutschland

Björn Berg
numetris AG
Essen
Deutschland

ISSN 1439-5428
ISBN 978-3-642-55063-8
ISBN 978-3-642-55064-5 (eBook)
DOI 10.1007/978-3-642-55064-5

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

Springer Vieweg

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier

Springer Vieweg ist eine Marke von Springer DE. Springer DE ist Teil der Fachverlagsgruppe Springer Science+Business Media
www.springer-vieweg.de

Vorwort

Die Hippiebewegung um 1960 war eine Protestbewegung gegen die Zwänge und bürgerlichen Tabus der etablierten Wohlstandsgesellschaft. Friedlicher Umgang, eine menschliche Lebensweise und Konsumkritik standen im Vordergrund dieser neuen Kultur.

„Respect people“ und „eliminate waste“ sind aber nicht nur Schlagworte der Flower-Power, sondern auch Prinzipien agiler Software-Entwicklungsmethoden. Während aber der einen Kultur der Einfluss halluzinogener Drogen nachgesagt wird, hat sich die andere inzwischen in zahlreichen Unternehmen etabliert.

Agile Softwareentwicklung wird gerne als Protestbewegung zu etablierten phasenorientierten Modellen verstanden und so treffen häufig die Vertreter agiler und phasenorientierter Methoden aufeinander und liefern sich mitunter erbitterte Glaubenskämpfe (anstelle funktionierender Software an den Kunden). Scheinen doch beide Welten so unvereinbar. Dabei haben doch beide Modellwelten das gleiche Ziel: Die pünktliche Lieferung der gewünschten Software an den Kunden unter Einhaltung der Kosten.

Wäre es darum nicht möglich, die Vorzüge beider Modelle zu kombinieren? Genau aus dieser Frage entstand die Idee zu einem hybriden Vorgehensmodell in der Softwareentwicklung!

Die Autoren stellen die Merkmale und Vorzüge der agilen und phasenorientierten Modelle detailliert dar und leiten daraus ihr hybrides Vorgehensmodell als Kombination beider Modelle ab. Und wie in der Pflanzenwelt präsentiert sich die Hybride mit einer hervorragenden Leistungsfähigkeit, was anhand zweier Fallstudien belegt wird.

Düsseldorf, November 2014 Prof. Dr. Gregor Sandhaus

Inhaltsverzeichnis

1 Einleitung 1
1.1 Ziel des Buches 4
1.2 Aufbau und methodisches Vorgehen 4
1.3 Leserkreis 5

2 Zentrale Begriffe 7
2.1 IT-Vorhaben 7
2.2 Stakeholder 8
2.3 Geläufige Umfänge für Entwicklungsvorhaben 8
2.4 Projektdimensionen 9
2.5 Arten von IT-Vorhaben 10
2.6 Prozessmodell und Vorgehensmodell 11
2.6.1 Prozess 11
2.6.2 Softwareentwicklungsprozess 12
2.7 Flexibilität von Prozessen 13
2.8 Durchlaufzeit von Prozessen 14
2.9 Die Prozesskostenrechnung 15
2.9.1 Verwendung der Prozesskostenrechnung in diesem Buch 16
2.9.2 Bestimmung der Prozesse 16
2.9.3 Festlegung der Prozessbezugsgrößen 17
2.9.4 Ermittlung von Prozesskostensätzen 18
2.10 IT-Compliance 18
2.11 ITIL™ 19
2.11.1 IT-Service-Lifecycle 20
2.11.2 Service Transition 21

3 Vorgehensmodelle für die Softwareentwicklung 23
3.1 Modellbegriff in der Informationstechnologie 24
3.2 Grundidee der Vorgehensmodelle/Prozessmodelle 25

3.3 Phasenorientierte Vorgehensmodelle 28
3.3.1 Wasserfallmodell 29
3.3.2 V-Modell 31
3.4 Agile Vorgehensweisen 33
3.4.1 Das agile Manifest 34
3.4.2 Lean Software Development 34
3.4.3 Extreme Programming 37
3.4.4 Scrum 39
3.5 Kritische Bewertung der Modelle 40
3.5.1 Struktur des Vorgehens 41
3.5.2 Praxisnähe 42
3.5.3 Skalierbarkeit 43
3.5.4 Vorgehen und Flexibilität 44
3.5.5 Unternehmenskultur 45
3.5.6 Methoden 46
3.5.7 Techniken und Werkzeuge 47
3.5.8 Unterstützende Prozesse 47
3.6 Zusammenfassung 50

4 Hybride Vorgehensmodelle 53
4.1 Kerngedanken des Modells 53
4.2 Anforderungen an das Modell 54
4.3 Mensch und Rollen 55
4.4 Grundlegendes Phasenmodell 56
4.5 Umgang mit äußeren Einflussfaktoren 58
4.5.1 Berücksichtigung von IT-Compliance 58
4.5.2 Einbindung von ITIL™ -Prozessen 59
4.6 Messung und Kennzahlen 60
4.6.1 Erfolge richtig messen 60
4.6.2 Kennzahlen für die Erfolgsmessung 61

5 Hybrides Vorgehen für Festpreisprojekte 63
5.1 Hybrides Phasenmodell 63
5.2 Methoden 66
5.2.1 Requirements Engineering 67
5.2.2 Kosten- und Aufwandsabschätzung 69
5.2.3 Nutzenanalyse 72
5.2.4 Prototyping und Feedbackmechanismen 74
5.2.5 Test-Driven Development 74
5.2.6 Acceptance Test-driven Development 76

5.2.7 Continuous Integration 77
5.2.8 Continuous Delivery 77
5.3 Techniken und Werkzeuge 78
5.4 Unterstützende Prozesse 79
5.4.1 Kontinuierlicher Verbesserungsprozess 79
5.4.2 Qualitätssicherung 80
5.4.3 Projektmanagement 82
5.4.4 Risikomanagement 82
5.5 Skalierbarkeit und Flexibilität 83
5.6 Unternehmenskultur 84
5.7 Besonderheiten bei der Bearbeitung von Kleinstvorhaben 85
5.7.1 Veränderungen am hybriden Phasenmodell 85
5.7.2 Methoden 86
5.7.3 Techniken und Werkzeuge 87
5.7.4 Unterstützende Prozesse 88
5.8 Zusammenfassung 89

6 Fallstudie numetris AG 91
6.1 Die numetris AG 92
6.2 Relevanz des hybriden Modells für die numetris AG 92
6.3 IST-Analyse 93
6.3.1 Genutzte Methoden 93
6.3.2 Ermittlung der Flexibilität 93
6.3.3 Ermittlung der Wertschöpfung 93
6.3.4 Einhaltung der IT-Compliance 95
6.4 Veränderungen durch das hybride Modell 96
6.4.1 Optimierung der genutzten Methoden 96
6.4.2 Erhöhung der Flexibilität 97
6.4.3 Ermittlung der Wertschöpfung 97
6.4.4 Einhaltung der IT-Compliance 98
6.5 Zusammenfassung und Ausblick 98

7 Fallstudie DEVK 101
7.1 Die DEVK Versicherungen 101
7.2 Relevanz der Thematik für die DEVK 102
7.3 Der Software-Wartungsprozess Auftragsbearbeitung 102
7.3.1 Ermittlung der Durchlaufzeit 103
7.3.2 Ermittlung der Flexibilität 103
7.3.3 Einhaltung von IT-Compliance 107
7.4 Veränderungen durch die Einführung des hybriden Modells 111

7.5 Erreichen der angestrebten Verbesserungen 111
7.5.1 Verkürzung der Durchlaufzeiten 112
7.5.2 Erhöhung der Flexibilität 112
7.5.3 Einhaltung von IT-Compliance-Anforderungen 114
7.6 Fazit 116

8 Zusammenfassung und Ausblick 117

Literatur 121

Abkürzungsverzeichnis

AE	Anwendungsentwicklung
AO	Anwendungsoptimierung
ASD	Adaptive Software Development
ATDD	Acceptance Test-driven Development
AUP	Agile Unified Process
BaFin	Bundesanstalt für Finanzdienstleistungsaufsicht
CASE	Computer-Aided Software Engineering
CD	Continuous Delivery
CFO	Chief Finance Officer
CI	Continuous Integration
CIO	Chief Information Officer
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integration
COBIT	Control Objectives for Information and Related Technology
COCOMO	Constructive Cost Model
COSMIC	Common Software Measurement International Consortium
DACH	Abkürzung für die Länder Deutschland, Österreich und Schweiz
ETXM	Akronym für Entry, Task, Exit, Measurement
FB	Fachbereich
FDD	Feature Driven Development
FP	Function Point
GoBS	Grundsätze ordnungsmäßiger DV-gestützter Buchführungssysteme
GQM	Goal Question Metrik
IB	Infrastruktur-Betrieb
ISBSG	International Software Benchmarking Standards Group
IE	Infrastruktur-Entwicklung
ISO	International Standardization Organization
ITIL™	Information Technology Infrastructure Library
KSt	Kostenstelle
KVP	Kontinuierlicher Verbesserungsprozess

LOC	Lines of Code
MaRisk	Mindestanforderungen an das Risikomanagement
NSN	Nokia Solutions and Networks; früher: Nokia Siemens Networks
OP	Object Point
OUP	Open Unified Process
PM	Personenmonat
PROFI	Professionelle Zusammenarbeit Fachbereich und IT
PT	Personentag
RE	Requirements Engineering
RFC	Request for Change
RUP	Rational Unified Process
SCM	Source-Code Management; Software Configuration Management
SOA	Service-orientierte Architektur
SPICE	Software Process Improvement and Capability Determination
TADP	Tideum Agile Development Process
TDD	Test Driven Development
TQM	Total Quality Management
UI	User Interface
UML	Unified Modelling Language
XP	Extreme Programming

Über die Autoren

Gregor Sandhaus ist Professor für Wirtschaftsinformatik an der Hochschule für Oekonomie & Management (FOM) in Essen. Mit über 25.000 Studierenden ist die gemeinnützige FOM die größte private Hochschule Deutschlands. Sie bietet Berufstätigen die Möglichkeit, sich parallel zum Job akademisch zu qualifizieren und staatlich wie international anerkannte Bachelor- und Master-Abschlüsse zu erwerben. Im Fokus der Lehre stehen praxisorientierte Studiengänge aus den Bereichen Wirtschaftswissenschaft und Ingenieurwesen.

Professor Sandhaus studierte Wirtschaftsingenieurwesen mit Schwerpunkt Informatik. Nach seiner Promotion zum Thema „Neural Networks for Cost Estimating in Project Management" war er zunächst als Seniorberater für IT-Strategien bei plenum, als Geschäftsführer bei entercom consulting GmbH und als Alliance Manager für SAP bei Sybase verantwortlich.

Seit seiner Berufung in 2008 lehrt Professor Sandhaus Fächer wie Softwareentwicklung, Projektmanagement und Datenbanken. Zusätzlich leitet er das Kompetenzfeld Softwareentwicklung am Institut für Logistik und Dienstleitungen und engagiert sich forschungsseitig für IT in der Logistik.

Nach seiner Ausbildung zum Fachinformatiker Systemintegration studierte **Björn Berg** Wirtschaftsinformatik und IT-Management an der Hochschule für Oekonomie und Management (FOM) in Essen. Sein Studium schloss er 2013 als Master of Arts ab.

Nach verschiedenen IT-Stationen beim Chemischen und Veterinäruntersuchungsamt Rhein-Ruhr-Wupper als Systemadministrator sowie beim Landesamt für Personaleinsatzmanagement NRW als Application Manager wechselte er 2009, zunächst als Software-Entwickler, zur heutigen numetris AG.

Seit 2011 ist der Wirtschaftsinformatiker als Vorstand des Essener Unternehmens vorrangig für die Bereiche Software-Entwicklung, Messdienstleistung und Marketing verantwortlich.

Philip Knott begann seine Ausbildung zum Fachinformatiker mit Schwerpunkt Anwendungsentwicklung im Jahr 2003 bei der Sparkassen Informatik. Im Anschluss daran begann er berufsbegleitend zu seiner Tätigkeit als Software-Entwickler ein Studium der Wirtschaftsinformatik an der Rheinischen Fachhochschule (RFH) in Köln.

Im Jahr 2008 wechselte er zur id Media GmbH. Dort sammelte er erste Erfahrungen in den Bereichen Projektleitung und Prozessoptimierung.

Nach dem Abschluss seines Studiums im Jahr 2010 wechselte er schließlich zu den DEVK Versicherungen, wo er bald die Leitung eines Software-Entwicklungsteams übernahm. Parallel zu dieser Tätigkeit absolvierte er von 2011 bis 2013 berufsbegleitend noch ein Studium zum Master of Arts IT-Management an der Hochschule für Oekonomie & Management (FOM).

Aktuell unterstützt der den IT-Leiter in einer Stabsfunktion bei zentralen Themen und der strategischen Weiterentwicklung der IT.

Abbildungsverzeichnis

Abb. 1.1 Phasen im Produktlebenszyklus 3

Abb. 2.1 Iron Projectmanagement Triangle 9
Abb. 2.2 Definition eines Prozesses 12
Abb. 2.3 Prozesshierarchie 17
Abb. 2.4 Rolle von Compliance im Unternehmen 19
Abb. 2.5 ITIL™ Kernpublikationen 20
Abb. 2.6 Die Phase der Service Transition im Überblick 21

Abb. 3.1 Verbreitung der Vorgehensmodelle (in Prozent) 24
Abb. 3.2 Spezifikation einer Teilaufgabe 25
Abb. 3.3 Ein einfaches Vorgehensmodell mit Rücksprüngen 26
Abb. 3.4 Überblick der Vorgehensmodelle zur Anwendungsentwicklung 27
Abb. 3.5 Verbreitung phasenorientierter Modelle (Angaben in Prozent) 28
Abb. 3.6 Erweitertes Wasserfallmodell nach Boehm 30
Abb. 3.7 Vereinfachte Darstellung des V-Modells 32
Abb. 3.8 Umfrageergebnis zum Einsatz agiler Methoden 34
Abb. 3.9 Das agile Manifest 35
Abb. 3.10 Scrum im Überblick 40
Abb. 3.11 Software Engineering-Strukturbaum 41

Abb. 4.1 Würfel der Beschränkungen 54
Abb. 4.2 Grundgerüst des hybriden Modells 56
Abb. 4.3 Agiles vs. phasenorientiertes Modell 57
Abb. 4.4 Zusammenspiel des hybriden Modell mit ITIL™ 59
Abb. 4.5 Angepasstes Modell des Erfolgs 61

Abb. 5.1 Hybrides Modell für projektorientierte Vorhaben 64
Abb. 5.2 Begriffe des agilen und klassischen RE und deren Zusammenspiel 68
Abb. 5.3 Quadrat des Teufels 69
Abb. 5.4 Ablauf des TDD-Zyklus 75
Abb. 5.5 Schematischer Ablauf von ATDD 76

Abb. 5.6 Zuordnung der Methoden und Werkzeuge 78
Abb. 5.7 Angepasstes Hybridmodell für Kleinstvorhaben 85

Abb. 6.1 Angepasstes Modell für die Wartung eines bestehenden Produkts 97
Abb. 6.2 Service-Transition im hybriden Modell 99

Abb. 7.1 Teilprozesse des Vorgehensmodells Auftragsbearbeitung 103
Abb. 7.2 Optimiertes Vorgehensmodell 111

Tabellenverzeichnis

Tab. 2.1 Klassifikation eines Vorhabens anhand der Codezeilen 8

Tab. 3.1 Übersicht der bewerteten Modelle..... 51

Tab. 5.1 Empfohlene Dauer einer Iteration anhand der Vorhabengröße 65

Tab. 6.1 Projektkalkulation für sequentielle Projekte 95
Tab. 6.2 Projektkalkulation für hybride Projekte 98

Tab. 7.1 Durchschnittliche Durchlaufzeiten für einen 1 PT-Auftrag..... 104
Tab. 7.2 Durchschnittlicher Aufwand für die definierten Teilprozesse..... 105
Tab. 7.3 Prozesskosten eines einfachen Durchlaufs 105
Tab. 7.4 Prozesskosten eines Durchlaufs mit ändernden Anforderungen 106
Tab. 7.5 Einhaltung von IT-Compliance im Prozess Auftragsbearbeitung..... 110
Tab. 7.6 Prozesskosten des optimierten Prozesses (einfacher Durchlauf)..... 113
Tab. 7.7 Prozesskosten des optimierten Prozesses (geänderte Anforderung)..... 114
Tab. 7.8 Erfüllung der IT-Compliance Anforderung im optimierten Prozess..... 115

Einleitung

1

Eine Ende Dezember 2012 durchgeführte Umfrage unter etwa 255 CFOs und CIOs hat gezeigt, dass ungefähr jedes sechste geschäftskritische IT-Projekt scheitert. Die von IDG Business Research Services durchgeführte Studie betrachtet dabei die Regionen DACH, USA/Kanada und Großbritannien.[1] In den konkreten Analysen der gescheiterten Projekte haben die Befragten angegeben, dass 46,8 % der Projekte sich verzögert haben, 11,8 % der Projekte teurer wurden und 8,1 % der Projekte nicht den erwarteten Nutzen gebracht haben. Die durchschnittliche Projektlaufzeit in der DACH-Region beträgt für die Durchführung 13 Monate. Dieser Zeit geht eine etwa sechsmonatige Planungsphase voraus.[2] Folglich ergibt sich eine durchschnittliche Projektlaufzeit von 19 Monaten.

Typisch bei längeren Projektlaufzeiten ist, dass sich „... die Aufgabenstellung von IT-Projekten... in der Regel während der Projektlaufzeit [verändert haben].“[3] Dies kann unter anderem an einer schlechten Projektvorbereitung oder an der Zugehörigkeit des Auftraggebers zu einer besonders dynamischen Branche liegen. Die Studie geht dabei nicht auf typische Konflikte zwischen Auftraggeber und Lieferant ein: Häufig werden IT-Leistungen in Form eines Werkvertrages an externe Lieferanten vergeben. Im Vorfeld werden alle Anforderungen ausgeschrieben und zum Festpreis beauftragt.[4] In der EU ist bspw. der öffentliche Dienst dazu verpflichtet, IT-Projekte europaweit auszuschreiben, wenn sie eine bestimmte Projektsumme übersteigen. Bei Unternehmen in regulierten Branchen ist

[1] Die regionalen Unterschiede spiegeln sich in der Projektlaufzeit und Dauer der Planungsphase wider. Auf die Faktoren für gescheiterte Projekte haben diese keine Auswirkung.

[2] Vgl. Quack (2013), S. 36 f.

[3] Kütz (2009), S. 11.

[4] Vgl. Henkel et al. (2011), S. 68.

Gregor Sandhaus et al., *Hybride Softwareentwicklung,* Xpert.press,
DOI 10.1007/978-3-642-55064-5_1

es ähnlich: Unternehmen vergeben Softwareprojekte als Werkvertrag. Dadurch sollen Geschäftsrisiken auf den Auftragnehmer übertragen werden.[5]

Für solche Projekte ist die Kalkulation der Projektkosten entscheidend, denn das Angebot regelt Termin und Preis. Um eine tragfähige Aufwands- und Kostenschätzung betreiben zu können, sind im Vorfeld zahlreiche Informationen erforderlich. Viele Phasenkonzepte berücksichtigen diesen Aspekt nicht. Erst im Laufe des Projekts werden die vollständigen Anforderungen in Form des Pflichtenhefts deutlich. Zu diesem Zeitpunkt ist in den Phasenmodellen der Vertrag zwischen Auftragnehmer und Auftraggeber aber schon zustande gekommen. Für Festpreisprojekte ist folglich eine ausführliche Anforderungsspezifikation schon vor Projektstart mit allen funktionalen und nicht-funktionalen Anforderungen notwendig.[6]

Die in der IT-Branche derzeit populären Methoden der agilen Vorgehensweisen können in einem Festpreisprojekt keine Abhilfe schaffen. Vom Wesen her sind sie sogar mit Festpreisprojekten unvereinbar. Der Entwicklungsprozess beginnt mit wenig konkreten Vorstellungen der Stakeholder über das zu entwickelnde System und anschließend hangelt sich das Entwicklungsteam von User-Story zu User-Story. Diese ziellose Ungewissheit wird von der Tatsache übertroffen, dass sich die Kosten für derartige Projekte im Vorfeld kaum abschätzen lassen.

Aber nicht nur die Art des Vorgehens zur Realisierung einer Softwarelösung sorgen dafür, dass sich Kosten für ein Projekt im Vorfeld kaum abschätzen lassen: Algorithmische und analoge Aufwandsschätzmethoden basieren auf empirisch ermittelten Produktivitätsdaten. Aus diesem Grund ist es schwierig, auf einer neuen Technologie basierende Projekte korrekt abzuschätzen. Die Produktivitätsdaten für diese Technologie sollten neu erhoben werden, um weitere Projekte akkurat kalkulieren zu können.[7]

Sneed hat 2004 in seinem Aufsatz *Jenseits vom IT-Projektmanagement* dargestellt, dass sich Softwaresysteme nicht nach den Maßgaben des klassischen IT-Projektmanagements realisieren lassen. Während Projektbeteiligte darauf hinarbeiten, erfolgreich das Ziel zu meistern, verändert sich gleichzeitig das Ziel.[8] „Schon allein das Erreichen des Ziels verändert die Bedingungen, unter denen das Ziel angestrebt wurde."[9] Dieses als Heisenberg-Prinzip bekannte Phänomen bewirkt, dass die Stakeholder bei kaum einem IT-Projekt mit den Ergebnissen zufrieden sind. Der Begriff IT-Projekt impliziert, dass am Ende eine endgültige Lösung existiert. Für die typischen betriebswirtschaftlichen Systeme trifft dies weniger zu. Extrinsische Einflüsse, wie gesetzliche Vorgaben und Änderungen in den betrieblichen Abläufen, erfordern häufige Änderungen am Produkt.[10]

---

[5] Vgl. ebd.; siehe auch Sneed (2007a).

[6] Vgl. Sneed (2007a), S. 55.

[7] Vgl. Sneed (2011), S. 81.

[8] Vgl. Sneed (2004), S. 33.

[9] Ebd.

[10] Sneed (2004), S. 33–34 und 37.

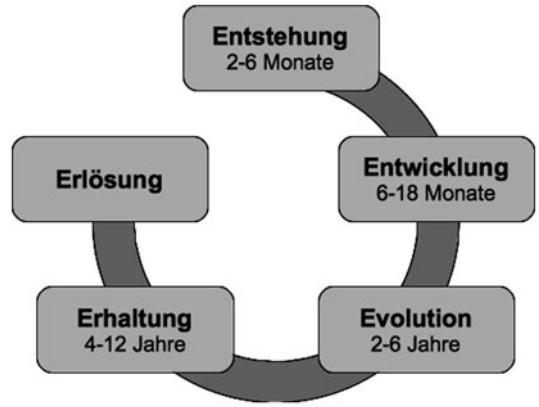

Abb. 1.1 Phasen im Produktlebenszyklus. (In Anlehnung an: Sneed (2004), S. 39)

Es ist daher sinnvoll, die Prozesse um ein Produkt statt um ein Projekt zu organisieren. Im Mittelpunkt steht ein Produkt, das evolutionär durch viele kleine Projekte weiterentwickelt wird. Der klassische Produktlebenszyklus aus der Betriebswirtschaftslehre kann somit auch auf den Lebenszyklus eines Software-Produkts angewendet werden (vgl. Abb. 1.1). Anstatt von einem endgültigen Produkt, wird von einem Release gesprochen. Jedes Release bezeichnet das Ende eines kleinen Projekts, das die Funktionalität der Software erweitert hat.[11] Sneeds Ergebnisse weichen somit nicht von den Ideen der Agilisten ab, die dieses Problem schon früh erkannt haben.

Agile Vorgehensweisen versuchen, dies in Form von produktiven Entwicklungszyklen von wenigen Tagen abzubilden. Am Ende eines Zyklus steht mindestens ein Prototyp mit ausgewählten Anforderungen des Kunden. Ein Festpreisprojekt lässt sich ebenfalls in kleine Phasen herunterbrechen. An die Einführungsphase schließt sich eine Wartungsphase an, sodass die Lehre des Produktlebenszyklus i. w. S. auch auf Festpreisprojekte angewendet werden kann.[12]

Wie in Abb. 1.1 zu sehen, ist nach Sneed die *Erhaltung* die längste Phase des gesamten Produktlebenszyklus. Somit spielt die Wartung einer Software eine zentrale Rolle im Entwicklungsprozess, weil sich viele Unternehmen schnell auf neue Marktentwicklungen einzustellen haben. Zum Beispiel werden kurzfristige Anpassungen der Geschäftsprozesse durchgeführt, um sich vom Wettbewerb abzuheben oder um gesetzliche Vorgaben einzuhalten.[13]

Um flexibel auf neue Marktsituationen reagieren oder neue Ideen schnell umsetzen zu können, ist es erforderlich, dass die IT-Abteilung eines Unternehmens zeitnah die Systeme

[11] Vgl. ebd.

[12] Die verschiedenen Projektarten werden zum Verständnis in Kap. 2.5 dargestellt.

[13] Vgl. Hofmann (2010), S. 230.

anpassen kann. Eines der vorherrschenden Ziele in der IT ist darum, neue Anforderungen mit minimaler Durchlaufzeit umzusetzen.[14]

1.1 Ziel des Buches

Im Rahmen dieses Buches soll überprüft werden, ob sich agile und phasengetriebene Vorgehensweisen kombinieren und auf Festpreisprojekte anwenden lassen. In diesem hybriden Modell werden die Phasen so angeordnet, dass die Angebotsphase stärker berücksichtigt wird, um dem Lieferanten eine verlässliche Kosten- und Aufwandsschätzung zu ermöglichen. Weiterhin erlaubt das Modell eine zeitliche Planung, in der sich Analyse, Implementierung und Auslieferung einem vorgegebenen Zeitrahmen unterordnen.

In der Projektsteuerung sind Methoden verankert, die einen gezielten Abbruch des Projekts erlauben, wenn die geforderte Zeit, die geplanten Kosten oder die gewünschte Qualität nicht eingehalten werden.

Schließlich wird aufgezeigt, wie das hybride Modell auch in der Softwarewartung eingesetzt werden kann. Hierbei wird berücksichtigt, dass die Anforderungsänderungen an eine bestehende Software sehr kleinteilig sein können.

1.2 Aufbau und methodisches Vorgehen

Zunächst werden die theoretischen Grundlagen für das Verständnis gelegt. Die Grundlagen sind in zwei Teile gegliedert: Im ersten Teil (Kap. 2, ab Abschn. 2.1) werden zentrale Begriffe definiert, die im weiteren Verlauf häufig genutzt werden. Ebenfalls wird die genutzte Quellenlage bewertet und Informationen zur zugrunde liegenden Literaturrecherche gegeben. Im zweiten Teil ab Kap. 3 wird zunächst der Modellbegriff in der Informatik vorgestellt und anschließend auf die Grundidee der Vorgehensmodelle eingegangen und typische Klassifizierungen vorgenommen. Die Auswahl der Methoden und Vorgehen basiert auf einer Umfrage unter Softwareentwicklern aus dem Jahr 2011. Da heutzutage in der Softwareentwicklung hauptsächlich zwischen agilen und phasenorientierten Methoden unterschieden wird, sind diese in Kap. 3, Abschn. 3.3 und 3.4, besonders hervorgehoben. Weiterhin werden Kriterien zur Bewertung der Modelle entwickelt, die aufzeigen, dass sich agile und phasenorientierte Vorgehensweisen weniger unterscheiden, als häufig angenommen wird.

Es werden Einflussfaktoren auf Softwareentwicklungsprozesse betrachtet, die sich inzwischen in den Vorgehensweisen etabliert haben (z. B. ITIL™) und abschließend gesetzliche Anforderungen untersucht, die häufig im Softwareprojekt auftreten.

[14] Vgl. Friedrichsen (2012), S. 14 ff.

Die aus den theoretischen Grundlagen gewonnenen Erkenntnisse haben Einfluss auf die Entwicklung des hybriden Modells in Kap. 4. Hierzu wird der grundsätzliche Ablauf erläutert und Methoden, Techniken, Werkzeuge und Prozesse genannt, die das Arbeiten mit einem hybriden Modell unterstützen.

Um die Praxistauglichkeit zu belegen, zeigen zwei Fallstudien, wie das hybride Modell eingesetzt werden kann. Die erste Fallstudie betrachtet die Einführung des hybriden Modells in der numetris AG (Kap. 6, ab Abschn. 6.1), die das hybride Modell zur Bearbeitung von Festpreisprojekten und für die Wartungsphase der eigenen Produkte einsetzt. In der zweiten Fallstudie (Kap. 7, Abschn. 7.1–7.6) wird am Beispiel der DEVK Versicherungen untersucht, wie das hybride Modell einen bestehenden Software-Wartungsprozess hinsichtlich Flexibilität und Durchlaufzeit verbessern kann.

Abschließend wird die Praxistauglichkeit des hybriden Modells evaluiert und Vorschläge unterbreitet, wie sich ein hybrides Vorgehensmodell in Organisationen etablieren lässt.

1.3 Leserkreis

Das vorliegende Buch hat einen starken Bezug zur Softwareentwicklung und zum Projektmanagement – ist aber nicht auf eine spezielle Branche beschränkt. Softwareentwicklung und Projektmanagement betreffen Softwareentwickler, Projektleiter, Führungskräfte und Entwicklungsteams. Aus diesem Grund wendet sich dieses Buch gleichermaßen an Unternehmensentscheider, Softwareentwickler, Projektleiter und Auftraggeber.

Projektleiter und Unternehmensentscheider finden in diesem Buch Hinweise auf die Gründe für das Scheitern von Projekten. Es wird nach konzeptionellen Lösungen gesucht, um das Scheitern von Softwareprojekten zu vermeiden. Ebenfalls werden neben klassischen Methoden zur Aufwandsschätzung und Nutzenanalyse exotischere Ansätze vorgestellt. Die vorgeschlagenen Metriken dienen dazu, ein Projekt frühzeitig gezielt abzubrechen, wenn sich herausstellt, dass der erwartete Nutzen in Zukunft nicht erfüllt werden kann.

Softwareentwickler finden ab Kap. 5 Methoden und Werkzeuge, die zur täglichen Arbeitsorganisation eingesetzt werden. Diese Methoden und Werkzeuge lassen sich in allen Phasen des Softwarelebenszyklus anwenden.

Zentrale Begriffe

2

Für das Verständnis des Buchs werden zunächst einige Begriffe definiert. Auf diese zentralen Begriffe wird im weiteren Verlauf immer wieder zurückgegriffen. Einige Definitionen zeigen neue Aspekte, die zu anderen Denkweisen in der Softwareentwicklung anregen sollen.

2.1 IT-Vorhaben

Das substantivierte Verb *vorhaben* beschreibt, dass „etwas verwirklicht werden soll“. In der einschlägigen deutschen Literatur zu den Themen Projektmanagement und Software Engineering wird der Begriff Vorhaben häufig mit Projekt gleichgesetzt. Korrekt ist, dass die Normenreihe DIN 69901 ein Projekt als Vorhaben beschreibt, dass sich durch die Einmaligkeit seiner Bedingungen auszeichnet. Diese Bedingungen werden als zeitliche, finanzielle und personelle Beschränkungen angesehen.[1]

Es gibt allerdings auch Vorhaben in der Softwareentwicklung, die durch wiederkehrende Tätigkeiten gekennzeichnet sind. Dazu gehört beispielsweise das Customizing von Formularen oder die Anpassung von Verfahren an neue gesetzliche Vorgaben.

Dieses Buch nutzt den Begriff IT-Vorhaben stellvertretend für Projekte in der Softwareentwicklung, aber auch für Kleinstmaßnahmen im täglichen Geschäftsablauf. Beide Arten unterliegen beim Vorgehen identischen Prozessschritten.

Darum werden die Begriffe Vorhaben und IT-Vorhaben synonym verwendet.

[1] Vgl. Kütz (2009).

Gregor Sandhaus et al., *Hybride Softwareentwicklung,* Xpert.press,
DOI 10.1007/978-3-642-55064-5_2

Tab. 2.1 Klassifikation eines Vorhabens anhand der Codezeilen

Umfang (LOC)	Codezeilen (pro PM)	Aufwand (in PM)	Klassifikation
10.000	2000–25.000	< 1	Klein
100.000	1000–20.000	5	Mittel
1000.000	700–10.000	100	Groß
10.000.000	300–5000	2000	Sehr groß

2.2 Stakeholder

Im klassischen Verständnis der Betriebswirtschaftslehre setzt sich die Gruppe der Stakeholder einer Organisation aus Arbeitnehmern, Lieferanten, Kunden und der Öffentlichkeit zusammen, also aus allen internen und externen Personengruppen, die am Handeln der Organisation direkt oder indirekt ein Interesse haben[2]. Im Kontext eines IT-Vorhabens verhält sich der Stakeholder-Ansatz genauso. In einem IT-Vorhaben gehören zu den Stakeholdern alle internen und externen Personengruppen, die von dem Vorhaben direkt oder indirekt betroffen sind. Dazu gehören bspw. verschiedene Fachabteilungen, die Informationstechnik (IT), die Unternehmensleitung, das Entwicklungsteam oder die Projektleitung.

2.3 Geläufige Umfänge für Entwicklungsvorhaben

In der Literatur werden kleine, mittlere und große Entwicklungsvorhaben unterschieden. Die Größenangabe für ein Entwicklungsvorhaben ist eine abstrakte Größe, denn sie wird durch verschiedene Indikatoren bestimmt. Dies können Umsatz, Personentage (PT) oder Personenmonate (PM), Dauer, Anzahl Codezeilen u. v. a sein. Interessant ist hierbei die Betrachtung der Entwicklungsvorhaben anhand der Codezeilen. Der Laie geht davon aus, dass ein Vorhaben mit 10-mal mehr Codezeilen auch 10-mal mehr Aufwand bedeutet. Tatsächlich nimmt der Aufwand eher exponentiell zu. Größere Vorhaben benötigen nämlich mehr Koordination zwischen größeren Personengruppen, womit gleichzeitig der Kommunikationsaufwand steigt.[3]

In der Softwareentwicklung herrscht somit der sog. Größennachteil vor: „. . . [Je] größer ein System wird, desto größer werden auch die Kosten je Einheit.“[4] McConnell[5] hat in

[2] Vgl. Wöhe und Döring (2008), S. 29 und S. 55.

[3] Vgl. Brooks (1995), S. 13–26; vgl. McConnell (2006), S. 91.

[4] Vgl. ebd.

[5] Vgl. McConnell (2006), S. 94.

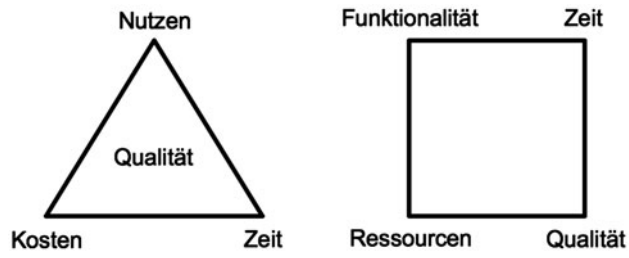

Abb. 2.1 Iron Projectmanagement Triangle. (In Anlehnung an: Appelo (2011), S. 185)

einer Untersuchung die generelle Beziehung zwischen der Größe eines Vorhabens und der Produktivität dargestellt. Daraus kann in Kombination mit den Ausführungen von Blum und Wörsdörfer[6] die in Tab. 2.1 pauschale Klassifizierung der Größe abgeleitet werden. Die Klassifizierung findet unabhängig von der Größe des Unternehmens und der Anzahl der verfügbaren Mitarbeiter statt, um so eine neutrale Bewertung anhand der Codebasis zu erstellen.

2.4 Projektdimensionen

Im Bereich des Projektmanagements wird häufig der Begriff *Iron Triangle* genutzt. Dieses Konzept zeigt, in welchen Dimensionen bzw. Grenzen sich ein Projekt bewegt. Die drei Größen oder Dimensionen Zeit, Kosten und Nutzen stehen in Konflikt zueinander und haben direkten Einfluss auf die daraus resultierende Größe Qualität.

Im Folgenden wird ein modifiziertes Konzept nach Appelo verwendet (vgl. Abb. 2.1). Das Dreieck wird zu einem Quadrat erweitert und der Qualitätsaspekt als eine direkt beeinflussbare Größe berücksichtigt. Dadurch sollen die in Kap. 3.4 ff. vorgestellten agilen Methoden in den Fokus gerückt werden, bei denen Qualität eine besondere Rolle spielt. Die Grundidee hinter dem veränderten Gedankenmodell: Verändert sich eine Größe in einer Ecke, so hat dies Einfluss auf die anderen Ecke. Zum Beispiel benötigen zusätzliche Funktionen entweder mehr Zeit oder mehr Ressourcen oder bewirken eine niedrigere Qualität. Ein Verlust von Ressourcen führt zu weniger Funktionen, einer geringeren Qualität oder einem verlängerten Zeitplan.[7] Dieses erweiterte Modell zur Darstellung der Dimensionen gilt auch für die Wartungsphase.

[6] Vgl. Blum und Worsdörfer (2013), S. 55; bezogen auf ein mittelständisches Unternehmen.
[7] Vgl. Appelo (2011), S. 185 f.

2.5 Arten von IT-Vorhaben

In der Softwareentwicklung gibt es nicht nur eine Art von IT-Vorhaben. Das Buch soll zeigen, dass sich für die nachfolgend gelisteten Arten von Vorhaben ein hybrides Modell eignet[8]:

- **Neu-Entwicklung:** In einem Projekt zur Neu-Entwicklung geht es um die Schaffung von Programmen und Datenbanken. Es ist das klassische Projekt in der Softwareentwicklung und wird auch häufig als Individualentwicklung bezeichnet.
- **Wartung:** Ziel von IT-Vorhaben in der Wartungsphase ist die kontinuierliche Weiterentwicklung bestehender Anwendungen und Systeme.
- **Migration:** Bei Vorhaben zur Migration werden Programme und Datenbanken von einer Umgebung in eine andere versetzt. Das Ziel ist erreicht, wenn die Anwendungen mit gleicher Funktionalität in der neuen Umgebung laufen.
- **Sanierung:** Werden Programme und Datenbanken in der gleichen Umgebung in einen besseren Zustand versetzt, so wird diese Art des Vorhabens als Sanierungs- bzw. Reengineeringprojekt bezeichnet.
- **Einführung:** Soll Standardsoftware in einer Organisation eingeführt werden, so wird dies als Einführungsprojekt bezeichnet. Die Techniken und Methoden beschränken sich auf die Migration der alten Daten und die Schaffung von Schnittstellen zu benachbarten Systemen.
- **Integration:** Ziel eines Integrationsvorhabens ist, bestehende Anwendungssysteme miteinander zu verknüpfen. Läuft eine Transaktion ohne Medienbruch über alle Systeme hinweg, so ist das Ziel des Projekts erreicht.

In der Praxis sind Sanierung- oder Migrationsprojekte eher selten. Migrationen sind typische Herausforderungen von Herstellern einer Standardsoftware, denn Migrationen werden meist dann durchgeführt, wenn der Anwender sich von bestehender Hardware oder Software trennt. Sanierungsprojekte waren in der Softwareindustrie nie sonderlich populär, da sich ihr Nutzen kaum quantifizieren lässt[9]. Integrationsprojekte sind so komplex, dass sie sich in drei Projekte teilen lassen: ein Aufklärungsprojekt, ein Realisierungsprojekt und ein Evolutionsprojekt. Diese Unterprojekte nutzen teilweise eigene Vorgehensmodelle (siehe Kap. 3).

[8] Vgl. Kütz (2009), S. 299; vgl. Sneed (2004); vgl. Sneed (2007c), S. 59–60.

[9] Vgl. Sneed (2007c), S.60.

2.6 Prozessmodell und Vorgehensmodell

In der Literatur wird häufig als Abgrenzung zwischen Prozess- und Vorgehensmodell angeführt, dass Vorgehensmodelle keine Aussagen zur Organisation und zu den Verantwortlichkeiten treffen. Diese Abgrenzung zwischen Vorgehensmodell und Prozessmodell wird in den nachfolgenden Ausführungen nicht vorgenommen, da im Hybriden Vorgehensmodell auch Empfehlungen zur Organisation genannt sind. Für die in der Fallstudie zur DEVK genutzten Prozesskostenrechnung wird jedoch zwischen Prozess und Prozessmodell unterschieden.

2.6.1 Prozess

Für Becker ist ein „Prozess .. die inhaltlich abgeschlossene, zeitliche und sachlogische Folge von Aktivitäten, die zur Bearbeitung eines betriebswirtschaftlich relevanten Objektes notwendig sind.“[10] Bei Womack handelt es sich bei einem Prozess um eine „Reihe von einzelnen Operationen, die zur Entwicklung einer Konstruktion, zur Erledigung eines Auftrags oder der Herstellung eines Produkts benötigt werden.“[11]

Auch bei Schweitzer findet man eine ähnliche Definition: „Ein Prozess ist dadurch gekennzeichnet, dass er eine Folge von Aktivitäten (Vorgängen, Tätigkeiten, Arbeitsgängen) umfasst, die sich auf ein bestimmtes Arbeitsobjekt beziehen und bei erneutem Arbeitsvollzug an einem Arbeitsobjekt identisch wiederholt werden“[12] Neu an dieser Definition ist der in den anderen Definitionen fehlende Aspekt, dass ein Prozess identisch wiederholt wird.

Eine sehr detaillierte Definition findet man bei Hoffmann: „Formal wird jeder Prozess über eine Reihe von Aktionen definiert, die aus einer gegebenen Menge von Eingangsgrößen schrittweise eine Menge von Ausgangsgrößen bilden. . . Für die Durchführung einer einzelnen Aktion werden unterschiedliche Ressourcen benötigt, die sich auf der obersten Ebene in drei Kategorien einteilen lassen. Der ersten Kategorie werden alle produktspezifischen Ressourcen wie Quelltexte und Planungsdokumente zugeordnet. Die zweite Kategorie umfasst die Software- und Hardware-Infrastruktur und die dritte Kategorie wird durch das zur Verfügung stehende Personal gebildet.“[13] Abbildung 2.2 zeigt die wesentlichen Merkmale der Definition in einer schematischen Übersicht.

Gemeinsam haben alle genannten Definitionen die folgenden Merkmale:

- Eine Abfolge von Aktivitäten
- Veränderung eines Objekts oder Erstellung einer Leistung
- Definierten Input und definierten Output

[10] Becker (2005), S. 6.
[11] Womack (2013), S. 408.
[12] Schweitzer (2008) S. 353.
[13] Hoffmann (2013), S. 491.

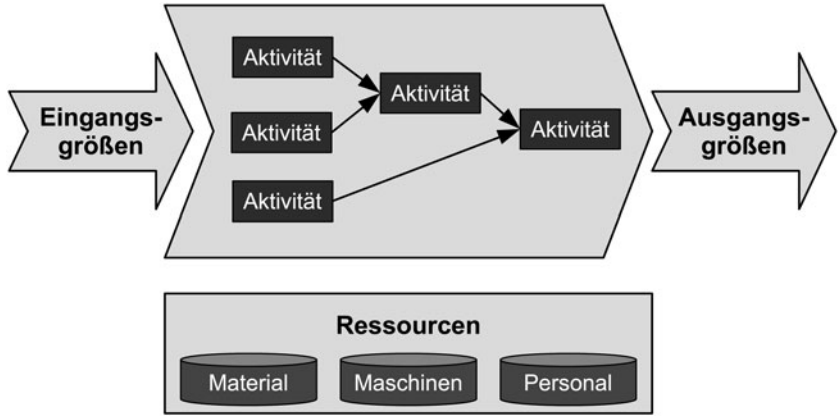

Abb. 2.2 Definition eines Prozesses. (In Anlehnung an: Hofmann (2013), S. 492)

Da die Hoffmann'sche Definition alle Aspekte der anderen abdeckt und teilweise um weitere wesentliche Aspekte ergänzt, wird im Folgenden diese als geltende Definition verwendet.

2.6.2 Softwareentwicklungsprozess

In der englischen Literatur werden die Begriffe „software development process" und „software process" häufig synonym verwendet.[14] In diesem Buch wird dies ebenfalls getan.

Sommerville definiert einen Softwareentwicklungsprozess als „a set of activities that leads to the production of a softwareproduct."[15] – also als eine Menge von Tätigkeiten, die zur Entwicklung eines Softwareprodukts führt. Nach seiner Aussage umfasst dies neben Erstellen einer völlig neuen Software auch das Erweitern und Anpassen von bestehenden Systemen und die Konfiguration und Integration von Standardsoftware oder Systemkomponenten.[16]

Münch definiert einen Softwareentwicklungsprozess hingegen als „a goal-oriented activity in the context of engineering-style software development."[17]

Bei O'Reagan findet sich die folgende Definition: „A software process is a set of activities, methods, practices, and transformations that people use to develop and maintain the software and the associated work products."[18]

Neben Aktivitäten machen für ihn also auch Methoden, Praktiken und Transformationen einen Softwareentwicklungsprozess aus. Auch er betont, dass sowohl die Erstellung als

[14] Münch (2012), S. 8.

[15] Sommerville (2007), S. 64.

[16] Vgl. ebd.

[17] Münch (2012), S. 8.

[18] O'Regan (2011), S. 3.

auch die Wartung von Software unter diese Definition fällt. Gleiches gilt für dazugehörige Arbeitsprodukte.

Da im Rahmen dieses Buches nicht nur Entwicklungsprojekte, sondern auch Prozesse, die zur Wartung bestehender Software genutzt werden, betrachtet werden und dabei auch die zu erstellende Dokumentation analysiert wird, gibt die Definition von O'Reagan alle wesentliche Aspekte wieder und wird daher im Folgenden verwendet.

Nach Sommerville gibt es den idealen Softwareentwicklungsprozess nicht. Daher haben viele Unternehmen ihre eigene Herangehensweise an die Softwareentwicklung erarbeitet.[19] Trotz vieler unterschiedlicher Softwareentwicklungsprozesse gibt es einige grundlegende Aktivitäten, die alle Softwareentwicklungsprozesse gemeinsam haben[20]:

1. **Spezifikation:** Die Funktionalität einer Software und ihre Grenzen werden definiert.
2. **Entwurf und Implementierung:** Die Software wird gemäß ihrer Spezifikation erstellt.
3. **Validation:** Die Software wird validiert, damit sichergestellt ist, dass sie den Kundenanforderungen entspricht.
4. **Evolution:** Die Software wird weiter entwickelt, um sich ändernde Kundenanforderungen zu erfüllen.

Auch bei O'Reagan findet sich eine vergleichbare Einschätzung: „The processes employed in software development include processes to determine the requirements; processes to design and develop the software; processes to verify that the software is fit for purpose; and processes to maintain the software."[21]

Wenn in diesem Buch der Begriff *Prozess* verwendet wird, ist im Allgemeinen ein Softwareentwicklungsprozess gemeint.

2.7 Flexibilität von Prozessen

Ein weiterer wesentlicher Begriff dieses Buches ist die Flexibilität. Kaluza definiert Flexibilität als „die Eigenschaft eines Systems proaktive oder reaktive sowie zielgerichtete Änderungen der Systemkonfiguration zu ermöglichen, um die Anforderungen von sich verändernden Umweltbedingungen zu erfüllen."[22] Diese allgemeine Definition bezieht sich nicht nur auf Prozesse, sondern ist allgemeingültig, da sie den Begriff System verwendet.

Für Lorenz bedeutet Flexibilität hingegen „situationsadäquat Abweichungen von Standards zu ermöglichen, seien es Abweichungen vom üblichen Fachprozess, der Einsatz

[19] Vgl. Sommerville (2007), S. 64.
[20] Vgl. ebd.
[21] O'Regan (2011), S. 3.
[22] Kaluza (2005), S. 9.

abweichender Technologie oder die Umsetzung mit abweichender Vorgehensweise.“[23] Die Definition von Lorenz umfasst zwar Prozesse, beschränkt sich jedoch nicht ausschließlich auf diese. Er fasst unter dem Begriff der Flexibilität auch den Einsatz von Technologien zusammen, die vom Standard abweichen.

Die Definition von Becker konzentriert sich hingegen auf die Flexibilität von Prozessen: „Flexibilität ist ein weiteres Kennzeichen guter Prozesse, denn flexible Prozesse können auf unterschiedliche oder geänderte Anforderungen zeitnah reagieren.“[24] Für ihn sichert die Flexibilität „nicht nur eine Reaktionsgeschwindigkeit, sondern ermöglicht auch, sich auf unterschiedliche Situationen einzustellen.“[25]

Die Analysen, die in den späteren Kapiteln dieses Buches durchgeführt werden, sind unabhängig von den eingesetzten Technologien und fokussieren sich ausschließlich auf einen Prozess. Daher wird in diesem Buch die Definition von Becker verwendet.

2.8 Durchlaufzeit von Prozessen

Best stellt den Begriff Durchlaufzeit sehr allgemein als „den gesamten Zeitbedarf dar – vom Start- bis zum Endpunkt des Prozesses.“[26]

Eine konkretere Definition findet man bei Womack. Er definiert die Durchlaufzeit als die „benötigte Zeit zwischen Konzept und Einführung, Auftrag und Auslieferung, oder vom Rohmaterial bis in die Hände des Kunden.“[27]

Da diese Definition nicht nur die Umsetzung, sondern auch die relevanten Aspekte der Erstellung von Konzepten und der Auslieferung beinhaltet, wird sie in diesem Buch als gültige Definition verwendet.

Sowohl Best[28] als auch Jammernegg[29] unterteilen die Durchlaufzeit in die Warte- und die Bearbeitungszeit. Als Bearbeitungszeit bezeichnet Best „ jene Zeitabschnitte der Durchlaufzeit, in der an der Lösung des Problems oder Fertigung des Produkts tatsächlich gearbeitet wird. Das Produkt wird sozusagen seiner Verkaufsreife näher gebracht.“[30] Das bedeutet, dass in den Bearbeitungszeiten wertschöpfende Aktivitäten stattfinden. Ihre Summe ist die Mindestdurchlaufzeit des Prozesses.[31]

[23] Lorenz (2012), S. 49.
[24] Becker (2008), S. 14.
[25] Ebd., S. 14.
[26] Best (2009), S. 79.
[27] Womack (2013), S. 405.
[28] Vgl. Best (2009), S. 80.
[29] Vgl. Jammernegg (2009), S. 220.
[30] Best (2009), S. 80.
[31] Vgl. Jammernegg (2009), S. 220.

Der zweite wesentliche Bestandteil der Durchlaufzeit ist die Wartezeit. „Wartezeiten ergeben sich immer dann, wenn der Vorgang unterbrochen wird, weil der jeweilige Mitarbeiter z. B. auf eine Information vom Kunden wartet oder das Produkt in der Fertigung bis zum nächsten Bearbeitungsschritt zwischengelagert wird, ohne dass dabei irgendein Fortschritt zu erkennen ist.“[32] Durch eine Reduzierung der Wartezeiten erreicht man kürzere Durchlaufzeiten. Dies führt nach Koch zu einer Erhöhung der Kundenzufriedenheit.[33] Nach Best entspricht die Summe aus Warte- und Bearbeitungszeit der Durchlaufzeit:

$$Durchlaufzeit = Wartezeit + Bearbeitungszeit$$

Formel 1: Berechnung der Durchlaufzeit
Vgl. Best (2009), S. 80

Daher ist es möglich, aus der Differenz von gemessenen Durchlauf- und Wartezeiten die Bearbeitungszeit zu berechnen.[34]

2.9 Die Prozesskostenrechnung

Die Prozesskostenrechnung ist eine Form der Kostenrechnung, die berücksichtigt, dass in vielen Unternehmen der Umfang an indirekten Leistungen, wie z. B. planende, steuernde und überwachende Tätigkeiten, zunimmt.[35]

Eine der Grundannahmen der Prozesskostenrechnung ist es, dass die Gemeinkosten der indirekten Leistungsbereiche nicht unmittelbar durch Produkte, sondern durch stellenübergreifende Prozesse verursacht werden.[36] Daher versucht die Prozesskostenrechnung, die Kosten aller betrieblichen Aktivitäten zu ermitteln und diese für die Kalkulation der Produkte nutzbar zu machen.[37]

Laut Jórasz sind die Ziele der Prozesskostenrechnung eine verursachungsgerechte Kalkulation, eine effiziente Planung und Kontrolle der Gemeinkosten sowie eine Erhöhung der Kostentransparenz in den indirekten Bereichen.[38] Sie kann sowohl auf Ist-, Normal- und/oder Plankosten basieren.[39]

[32] Best (2009), S. 80.
[33] Vgl. Koch (2011), S. 77.
[34] Best (2009), S. 80.
[35] Vgl. Jórasz (2008), S. 281.
[36] Vgl. Schweitzer (2008), S. 352 f.
[37] Vgl. Coenenberg (2012), S. 162.
[38] Vgl. Jórasz (2008), S. 281.
[39] Vgl. Freidank (2008), S. 371.

2.9.1 Verwendung der Prozesskostenrechnung in diesem Buch

In diesem Buch wird keine vollständige Prozesskostenrechnung durchgeführt. Um Ist- und Soll-Prozesse vergleichen zu können, ist die Bestimmung der Prozesskostensätze ausreichend. Ein Prozesskostensatz stellt nach Jórasz die durchschnittlichen Kosten einer einmaligen Durchführung eines Prozesses dar.[40] Sie können für Soll-/Ist-Vergleiche genutzt werden.[41]

Bei einer vollständigen Prozesskostenrechnung werden nach der Berechnung des Prozesskostensatzes die Kosten der leistungsmengenneutralen Prozesse auf die Prozesskostensätze verteilt. Dies geschieht nach dem Durchschnittsprinzip.[42] Die geplanten Prozessoptimierungen in diesem Buch beziehen sich ausschließlich auf Softwareentwicklungsprozesse und haben daher keine Auswirkungen auf die indirekten Bereiche. Daher sind die Kosten aus diesen Bereichen vor und nach der Optimierung gleich hoch und haben bei einem Vergleich der Ist- und Soll-Situation keine Bedeutung.

Die Ermittlung der Prozesskostensätze erfolgt nach Jórasz in drei Schritten[43]:

- Bestimmung der Prozesse
- Festlegung der Bezugsgrößen
- Bildung von Prozesskostensätzen

Jeder dieser Schritte wird in einem der nächsten Kapitel genauer erläutert.

2.9.2 Bestimmung der Prozesse

Für eine Prozesskostenrechnung ist eine intensive Tätigkeitsanalyse der betroffenen Kostenbereiche notwendig.[44] „Zunächst werden alle wesentlichen Tätigkeiten listenartig festgestellt. Dazu kann man auf Arbeitspläne und Stellenbeschreibungen zurückgreifen oder die notwendigen Informationen durch Interviews beschaffen.“[45]

Die ermittelten Teilprozesse werden anschließend kostenstellenübergreifend zu Hauptprozessen zusammengefasst. Die Hauptprozesse bilden dann die Grundlage für die Prozesskostenkalkulation.[46] Dieser Schritt ist für die Prozesskostenrechnung von zentraler Bedeutung, weil durch ihn die Grundlage geschaffen wird, die betrieblichen Aktivitäten als

[40] Vgl. Jórasz (2008), S. 285.
[41] Vgl. Schmidt (2008), S. 232.
[42] Vgl. Jórasz (2008), S. 285.
[43] Vgl. ebd., S. 282.
[44] Vgl. ebd., S. 283.
[45] Schmidt (2008), S. 224.
[46] Vgl. Jórasz (2008), S. 283.

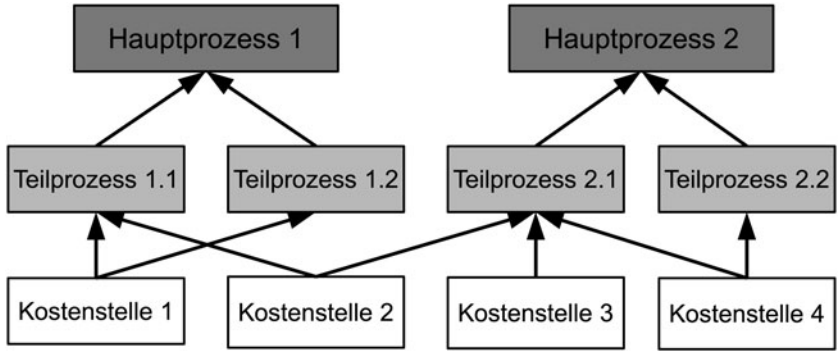

Abb. 2.3 Prozesshierarchie. (Eigene Darstellung in Anlehnung an Jórasz (2008), S. 283)

Kalkulationsobjekt zu benutzen.[47] Abbildung 2.3 stellt diese Prozesshierarchie schematisch dar.

„Die Teilprozesse sind daraufhin zu untersuchen, ob sie sich mit dem Leistungsvolumen verändern, oder ob sie mengenunabhängig sind und generell anfallen. Teilprozesse der erstgenannten Art werden leistungsmengeninduziert (lmi), die der letztgenannten Art werden als leistungsmengenneutral (lmn) bezeichnet."[48] Üblicherweise wird die Häufigkeit bzw. die Anzahl der Vorgänge (z. B. die Anzahl der Bestellungen) als Kostentreiber genutzt. Dies betrifft nur die leistungsinduzierten Prozesse. Die leistungsmengenneutralen Prozesse zeichnen sich hingegen dadurch aus, dass sich ihr Ergebnis nur schlecht quantifizieren lässt und kein direkter Bezug zum Leistungsprozess des Unternehmens besteht.[49]

2.9.3 Festlegung der Prozessbezugsgrößen

Im nächsten Schritt erfolgt die Zuordnung von Kapazitäten und Kosten zu den gebildeten Prozessen. „Die Häufigkeit, mit der ein Prozess in der Abrechnungsperiode wiederholt wird, ist die Prozessmenge."[50] „Die Maßgröße der Prozessmenge wird Prozessbezugsgröße (cost driver) genannt. Sollen die Gemeinkosten eines Prozesses verursachungsgerecht auf Kostenträger verteilt werden, muss gefordert werden, dass zwischen der auftretenden Kosteneinflussgröße und der definierten Prozessbezugsgröße des betrachteten Prozesses eine funktionale Beziehung besteht, so dass zwischen der Prozessbezugsgröße (Kosteneinflussgröße) und den Kosten des Prozesses eine funktionale Beziehung besteht."[51]

[47] Freidank (2008), S. 377 f.
[48] Schmidt (2008), S. 224.
[49] Vgl. ebd., S. 226.
[50] Schweitzer (2008), S. 355.
[51] Ebd.

2.9.4 Ermittlung von Prozesskostensätzen

Im nächsten Schritt werden Prozesskostensätze ermittelt. Nach Schweitzer wird ein Prozesskostensatz durch die Division der Prozesskosten durch die zugehörige Ausprägung der Prozessbezugsgröße ermittelt.[52]

$$Prozesskostensatz = \frac{Prozesskosten}{Prozessmenge}$$

Formel 2: Ermittlung des Prozesskostensatz
Coenenberg (2012), S. 171

2.10 IT-Compliance

Nach Gaulke bezeichnet IT-Compliance „die Einhaltung der relevanten Gesetze und Rechtsverordnungen, der vertraglichen Verpflichtungen sowie der externen und internen Richtlinien, die die IT einer Organisation betreffen."[53]

Bei Hansen findet man eine weitestgehend deckungsgleiche Definition: „IT-Compliance bezweckt die Einhaltung von gesetzlichen und vertraglichen Regelungen, von Richtlinien sowie von Berechtigungskonzepten im IT/IS Bereich."[54] Er ergänzt also auch die Einhaltung von Berechtigungskonzepten.

Für Laudon ist IT-Compliance ein Zustand, „in dem alle für die Unternehmens-IT relevanten Rechtsnormen (Gesetze und die damit zusammenhängenden Bestimmungen und Verordnungen) sowie Regelungswerke von Behörden zur Interpretation oder Ausführung dieser Rechtsnormen nachweislich eingehalten werden."[55] Darunter fallen für ihn auch Richtlinien und Selbstverpflichtungen.[56]

Alle drei Definitionen weisen zwei Merkmale auf:

- Einhaltung von relevanten Rechtsnormen
- Erfüllung von externen und internen Richtlinien der Organisation

Sowohl Gaulke als auch Hansen führen darüber hinaus die Einhaltung von vertraglichen Verpflichtungen als zentrale Merkmale von IT-Compliance auf. Die Anforderung nach einer Einhaltung von Berechtigungskonzepten findet man nur bei Hansen.

---

[52] Vgl. Schweitzer (2008), S. 356.
[53] Gaulke (2010), S. 183.
[54] Hansen (2009), S. 195.
[55] Laudon (2010), S. 877.
[56] Vgl. ebd., S. 877.

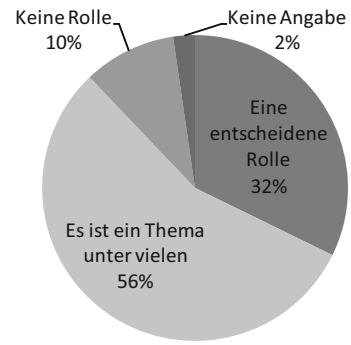

Abb. 2.4 Rolle von Compliance im Unternehmen. (Eigene Darstellung in Anlehnung an ELEVEN Security (2012))

Die Definition von Hansen beinhaltet alle wesentlichen Merkmale, die in den anderen Definitionen gefunden werden können und ergänzt diese um den wesentlichen Aspekt der Benutzerberechtigungen. Daher wird im Rahmen dieses Buches die Definition von Hansen genutzt.

Die Konformität mit Regelwerken ist im Laufe des letzten Jahrzehnts unter dem Einfluss gesetzlicher Vorgaben für viele Unternehmen ein wichtiges Thema geworden.[57] In einer im Herbst 2012 durchgeführten Umfrage des Unternehmens ELEVEN gaben ca. ein Drittel der befragten Unternehmen an, dass Compliance bei ihnen eine zentrale Rolle spielt (siehe Abb. 2.4).

Es werden unterschiedliche IT-Compliance-Anforderungen an Vorgehensmodelle der Softwareentwicklung gestellt. Diese können allgemeingültig oder branchenspezifisch sein. Im Rahmen der Fallstudien in den Kap. 6 und 7 werden konkrete IT-Compliance-Anforderungen genannt und deren Einhaltung im hybriden Modell überprüft.

2.11 ITIL™

„Die Information Technology Infrastructure Library™ (ITIL™) bietet eine systematische Einführung in die Förderung der Qualität von IT-Services.“[58] Seit vielen Jahren ist ITIL™ als de facto Standard für das IT-Service-Management etabliert.[59] Im Jahre 2007 wurde die aktuelle Version 3 (V3) veröffentlicht.[60]

Unter dem Begriff IT-Service-Management versteht Beims „die Steuerung aller fachlichen Fähigkeiten der Organisation zur Bereitstellung eines Mehrwertes für den Kunden in Form von Services.“[61] „IT-Service-Management bedeutet also, die Qualität und Quantität der IT-Services zu planen, zu überwachen und zu steuern.“[62]

[57] Vgl. Niemann (2005), S. 141.
[58] van Bon (2009), S. 15.
[59] Vgl. Beims (2010), S. 12.
[60] Vgl. Wischki (2009), S. 119.
[61] Beims (2010), S. 3.
[62] Ebd., S. 2.

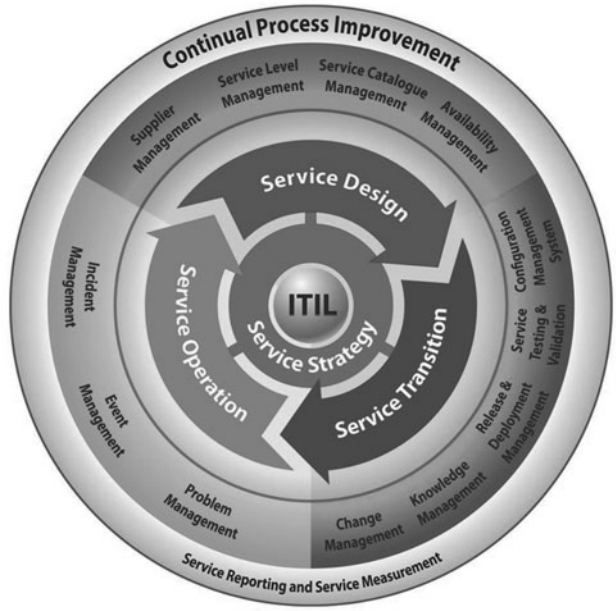

Abb. 2.5 ITIL™ Kernpublikationen. (Anderson (2009))

In der obigen Definition des IT-Service-Managements ist der Begriff IT-Service enthalten. Auch dieser ist für das Verständnis von ITIL™ wesentlich. „Ein Service liefert dem Kunden einen definierten Nutzen, ohne dass dieser für die spezifischen Risiken und Kosten der Serviceerbringung verantwortlich ist.“[63] Das Besondere an einem Service ist also, dass Kunden einen Nutzen zur Verfügung gestellt bekommen, ohne dass sie dafür Sorge zu tragen haben, auf welche Art und Weise dies geschieht.

ITIL™ versteht sich selbst als „Good Practice“. „Good Practice ist ein Ansatz oder eine Methode, die sich selbst in der Praxis bewährt hat. Good Practices können eine massive Unterstützung für Organisationen sein, die ihre IT-Services verbessern wollen.“[64] Die in ITIL™ vorgeschlagenen Praktiken beschreiben Erfahrungen und Bewährtes und legen sich nicht auf eine bestimmte Unternehmensform oder Branche fest.[65]

2.11.1 IT-Service-Lifecycle

In ITIL™ V3 gibt es einen IT-Service-Lifecycle, den alle Services durchlaufen. Dieser setzt sich aus den Bereichen Service Strategy, Service Design, Service Transition, Service Operation und Continual Service Improvement zusammen (siehe Abb. 2.5).

[63] Beims (2010), S. 3.
[64] van Bon (2009), S. 21.
[65] Vgl. Beims (2010), S. 11.

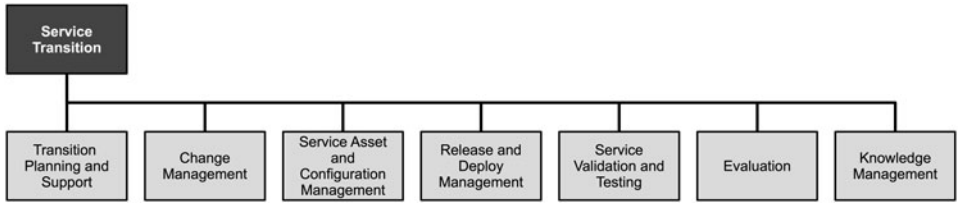

Abb. 2.6 Die Phase der Service Transition im Überblick. (Eigene Darstellung in Anlehnung an Wischki (2009), S. 141)

Im Rahmen dieses Buches werden nicht alle der fünf Bereiche im Detail dargestellt. Da sich dieses Buch auf die Verbesserung eines Softwareentwicklungsprozesses bezieht, werden lediglich zwei Prozesse der *Service Transition* näher betrachtet.

2.11.2 Service Transition

Die Service Transition stellt Prozesse und Verfahren zur Überführung von qualitätsgesicherten, neuentwickelten oder veränderten Services in den operativen Betrieb sowie zum Ausphasen bzw. Eliminieren von bestehenden IT-Services bereit.[66]

Da nach Beims technische und organisatorische Änderungen eine der häufigsten Fehlerquellen bei der Lieferung von Services sind,[67] haben die Prozesse der Service Transition einen maßgeblichen Anteil an der Verfügbarkeit der Services. Abbildung 2.6 listet alle Prozesse der Service Transition Phase auf:

Von den genannten Prozessen sind für die Softwareentwicklung vor allem die Prozesse *Change Management* und das *Release and Deploy Management* relevant und sind wie folgt definiert:

- **Change Management**: „Der Change-Management-Prozess gehört für den IT-Betrieb zu den wichtigsten und entscheidendsten Prozessen.“[68] Mit Hilfe des Prozesses wird sichergestellt, dass Änderungen kontrolliert durchgeführt werden. Das bedeutet, dass Bewertung, Priorisierung, Planung, Tests, Implementierung und Dokumentation der Änderungen stattgefunden haben.[69] Daher ist es notwendig, dass jede Änderung an einem IT-Service oder IT-System vorab den Change-Prozess durchläuft.[70] Initiiert wird der Change-Prozess über einen Request for Change (RFC). Nach Bon handelt es sich bei einem RFC um einen formalen Antrag, um ein oder mehrere Configuration Items zu

[66] Vgl. Wischki (2009), S. 141.
[67] Vgl. Beims (2010), S. 91 f.
[68] Wischki (2009), S. 31.
[69] Vgl. van Bon (2009), S. 44.
[70] Vgl. Wischki (200), S. 31.

ändern.[71] Alle Komponenten, z. B. IT-Services, Hardware, Software, Dokumentationen und Verträge, die von der IT-Infrastruktur verwaltet werden, fallen nach Beims unter den Begriff Configuration Item.[72] Ein RFC wird in der Regel in elektronischer Form bereitgestellt und ist die Basis für die Bewertung, Planung und Genehmigung eines Changes.[73] Die Art der geplanten Änderung ist für die Erstellung eines RFC völlig unerheblich. Sowohl die Änderung der Hintergrundfarbe einer Applikation als auch die Anpassung eines Konfigurationsparameters zur Fehlerbeseitigung sind über einen RFC zu beantragen.[74] Ein genehmigter Change wird im Prozess *Release and Deployment Management* bei der Release-Planung berücksichtigt.[75]

- **Release and Deployment Management**: Das Ziel des „Release and Deployment Management" ist es, Releases erfolgreich in die geplante Zielumgebung zu integrieren, ohne dass unvorhergesehene Auswirkungen auf bestehende Services auftreten.[76] Ein Release versteht man hierbei als „ein Set an neuen oder zu ändernden [Configuration Items], die zusammen getestet und in die Produktivumgebung ausgerollt werden."[77] Zu den wichtigsten Aufgaben des Release and Deployment Managements gehört das Entwickeln und Einführen von Release Policies. Diese mit den Kunden vereinbarten Regeln enthalten u. a. die Frequenz und den Typ der Releases.[78] Darüber hinaus erstellt das „Release and Deployment Management" auch verständliche Release- und Deployment-Pläne für die Abstimmung und Zusammenarbeit mit dem Change Management.[79]

[71] Vgl. van Bon (2009), S. 116.
[72] Vgl. Beims (2010), S. 104.
[73] Vgl. ebd., S. 98.
[74] Vgl. Wischki (2009), S. 31.
[75] Vgl. ebd., S. 39.
[76] Vgl. Beims (2010), S. 110.
[77] van Bon (2009), S. 124.
[78] Vgl. Wischki (2009), S. 51.
[79] Vgl. Beims (2010), S. 110.

Vorgehensmodelle für die Softwareentwicklung 3

Die Umsetzung von IT-Vorhaben sollte in einem Unternehmen nach einem standardisierten Ablauf erfolgen. Ein standardisierter Verlauf ist unabhängig vom Vorhaben und stellt sicher, dass die Aufgaben ohne Auslassung von Teilschritten ausgeführt und abgeschlossen werden[1]. „Üblicherweise werden [Vorhaben] nach allgemeinen Regeln – sogenannten Vorgehensmodellen – durchgeführt. Gleichwohl müssen diese Vorgehensmodelle. . . angepasst werden. . .“[2] Die Anpassung der Vorgehensmodelle geschieht nach Inhalt und Größe der Vorhaben und wird als Tailoring bezeichnet[3].

In der IT existieren mehrere dokumentierte und standardisierte Verläufe für IT-Vorhaben jeglicher Art. Einige davon sind eher wissenschaftlich, andere praxisnah erprobt. Im Folgenden wird zunächst die Grundidee hinter Vorgehensmodellen erläutert und anschließend werden die Vorgehensmodelle klassifiziert sowie die in der Praxis gebräuchlichsten Vorgehensmodelle vorgestellt. Im Rahmen einer Stärken-Schwächen-Analyse werden die Modelle abschließend bewertet.

Die Auswahl der ab Kap. 3.4 beschriebenen Modelle ist im Vorfeld auf Basis der Ergebnisse einer Umfrage zum Softwaretest der Hochschulen Bremen, Bremerhaven und Köln in Zusammenarbeit mit den Firmen ANECON, GTB und STB ausgewählt worden. In der Umfrage wurde die prozentuale Nutzung der verschiedenen etablierten Modelle in der Praxis ermittelt. Die Ergebnisse sind in Abb. 3.1 dargestellt. Dabei hat sich gezeigt, dass im Praxiseinsatz die phasenorientierten Modelle (vgl. Kap. 3.3) deutlich dominieren[4].

[1] Vgl. Wieczorrek und Mertens (2008), S. 53.

[2] Kütz (2009), S. 8 f.

[3] Vgl. Henrich (2002), S. 30 ff.; vgl. Kütz (2009), S. 9; vgl. Wieczorrek und Mertens (2008), S. 53.

[4] HS Bremen et al. (2011).

Gregor Sandhaus et al., *Hybride Softwareentwicklung,* Xpert.press,
DOI 10.1007/978-3-642-55064-5_3

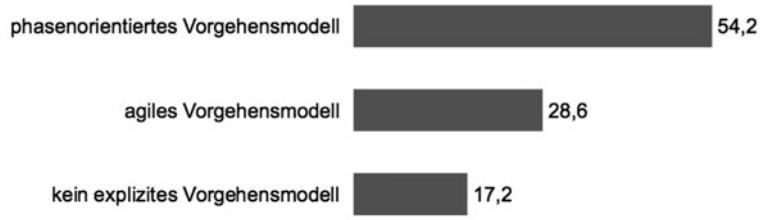

Abb. 3.1 Verbreitung der Vorgehensmodelle (in Prozent). (Entnommen von: HS Bremen et al. (2011))

3.1 Modellbegriff in der Informationstechnologie

Ursprünglich wurde der Begriff Modell in der bildenden Kunst verwendet. Dort fasst er die Form, Beschaffenheit und Maßverhältnisse eines vorhandenen oder noch zu schaffenden Gegenstandes in bestimmtem, besonders verkleinerndem Maßstab zusammen.

In der Betriebswirtschaftslehre helfen Modelle „die komplexen Zusammenhänge der wirtschaftlichen Wirklichkeit zu vereinfachen, um sie überschaubar zu machen und um am Modell zur Erkenntnis von Grundzusammenhängen und Prozessen zu gelangen. . . “[5] Modelle dienen also in der Betriebswirtschaftslehre zur Abstraktion der Realität und zur vereinfachten Abbildung von Prozessen. Zur Entwicklung von Annahmen für den Aufbau des Modells wird die Methode der Reduktion verwendet.

Hansen und Neumann definieren den Begriff Modell pragmatisch als „eine Abstraktion des betrachteten Realitätsausschnitts. Unter Modellierung werden die Tätigkeiten verstanden, die zur Definition eines Modells führen.“[6] Diese Eigenschaft hilft, mit Modellen gezielter kommunizieren zu können und durch bewusstes Weglassen Komplexität zu reduzieren und den Überblick zu behalten[7].

Krcmar gibt vier Elemente für die Modellbildung in der Informationstechnologie vor: die Abbildungsregeln, das Modellsubjekt, die abzubildende Realität und den Adressaten der Modellbetrachtung. Die Abbildungsregeln sind dabei mit der Reduktionsmethode aus der Betriebswirtschaftslehre gleichzusetzen. Sie schreiben vor, wie die Realität abzubilden ist. Das Modellsubjekt ist der Erzeuger eines Modells. Hierbei handelt es sich in den meisten Fällen um eine natürliche Person, die mit Hilfe der Abbildungsregeln die Realität in ein abstraktes Modell überführt. Der Adressat sollte unter Berücksichtigung der Abbildungsregeln in der Lage sein, das Modell zu verstehen.[8]

[5] Wöhe und Döring (2008), S. 12.

[6] Hansen und Neumann (2005), S. 174.

[7] Vgl. Ditze et al. (2013), S. 22.

[8] Vgl. Krcmar (2010), S. 21.

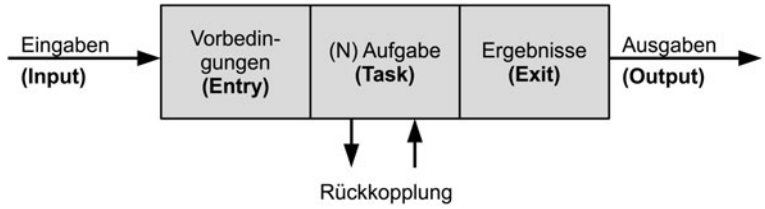

Abb. 3.2 Spezifikation einer Teilaufgabe. (Eigene Darstellung in Anlehnung an: Henrich (2002), S. 32)

Bezogen auf die Entwicklung von Software helfen Modelle, die Realität zu vereinfachen, die Komplexität zu reduzieren und den Blick auf wichtige Prozesse und Methoden zu fokussieren.

3.2 Grundidee der Vorgehensmodelle/Prozessmodelle

Das Ziel von Vorgehensmodellen ist die hierarchische Gliederung der Gesamtaufgabe. Die Darstellung der Gliederung erfolgt entweder zerlegungsorientiert (Baum oder Hierarchie) oder ablauforientiert (Graph). Die Teilphasen werden jeweils als Knoten repräsentiert. Bei der ablauforientierten Sicht werden die Abhängigkeiten der Phasen durch die Kanten dargestellt.[9]

Der Begriff Aufgabe beschreibt die durchzuführenden Tätigkeiten, während der Begriff Aktivität die Erfüllung oder Ausführung einer Aufgabe meint.

Humphrey hat 1989 die Definition einer Teilaufgabe (vgl. Abb. 3.2) in einem Vorgehensmodell nach der ETXM-Spezifikation (Entry, Task, Exit, Measurement) in die folgenden Bestandteile zerlegt[10]:

- **Vorbedingungen:** Die Bedingungen, die vor Durchführung der Aufgabe erfüllt sein sollten.
- **Ergebnisse:** Die Resultate, die erzeugt werden, und wie sie aussehen.
- **Rückkopplung:** Jede Rückkopplung von bzw. zu einer anderen Aufgabe.
- **Aufgabe:** Was ist zu tun, durch wen, wie und wann, einschließlich entsprechender Standards, Verfahren und Verantwortlichkeiten.
- **Maße:** Die geforderten Maße für die Aufgabe (Aktivitäten, Ressourcen, Zeit), für die Ergebnisse (Anzahl, Größe, Qualität) und für die Rückkopplungen (Anzahl, Größe, Qualität).

[9] Vgl. Henrich (2002), S. 30 f.

[10] Nach Humphrey (1989), S. 257 ff.; zitiert in: Henrich (2002), S. 32.

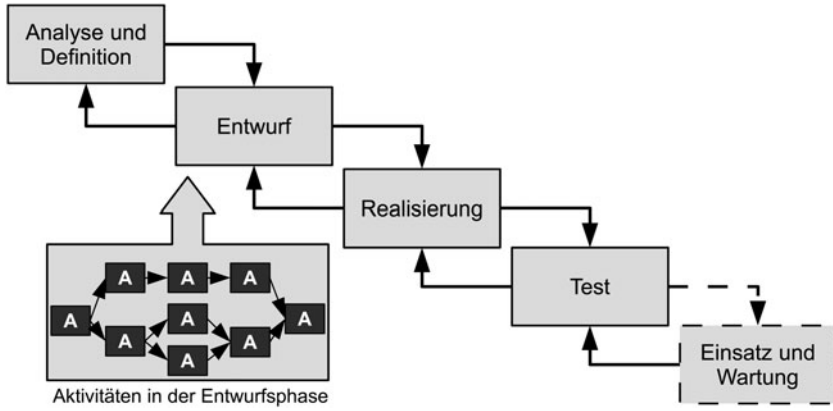

Abb. 3.3 Ein einfaches Vorgehensmodell mit Rücksprüngen. (Eigene Darstellung in Anlehnung an Henrich (2002), S. 38)

Aus der Darstellung und Beschreibung lassen sich wichtige Faktoren für eine Aktivität und die Mindestkriterien für ein Vorgehensmodell ableiten. In der Beschreibung zu einer Aufgabe sind die einzusetzenden Methoden zu erfassen (Wie), um die spätere Nachvollziehbarkeit zu fördern. Befolgt ein Unternehmen in allen IT-Vorhaben die gleichen Standards und Methoden, können neue Teammitglieder die bisherigen Arbeitsergebnisse besser verstehen. Die Einarbeitung wird so erleichtert.

Ein Vorhaben in der Anwendungsentwicklung durchläuft einen Zyklus mit Phasen. Dieser Zyklus wird häufig durch lineare Phasenmodelle dargestellt. Das Grundprinzip der Phasenmodelle ist immer gleich: „Nach der Definition der Systemanforderungen wird ein vorläufiges Konzept aufgestellt, das dann hinsichtlich der Anforderungen überprüft, entsprechend modifiziert und verfeinert wird. Nachdem der Entwurf erstellt bzw. das Design abgeschlossen ist, kann die Implementierung beginnen, die zuerst vorläufig ist und eine Phase des Testens zur Folge hat.“[11] Die Inbetriebnahme und Wartung des Systems bilden in allen Konzepten die letzten Phasen und treten erst ein, wenn bei der implementierten Lösung keine Fehler gefunden wurden und diese erfolgreich auf den Systemen installiert ist.

Vorgehensmodelle basieren in der Praxis auf Phasenkonzepten. Die einzelnen Phasen folgen sequentiell[12] aufeinander und werden dabei durch definierte Meilensteine voneinander abgegrenzt. In jeder Phase werden eine Menge von Aktivitäten durchgeführt, die wiederum sequentiell oder parallel bearbeitet werden (siehe Abb. 3.3).

Dieses Modell wird auch als Wasserfallmodell bezeichnet, auf das später noch detailliert eingegangen wird (siehe Kap. 3, Abschn. 3.3.1). Das Wasserfallmodell kann durch Rücksprünge in vorangegangene Phasen ergänzt werden, die dann erlaubt sind, wenn festgestellt wird, dass festgelegte Ziele sonst nicht erreicht werden können.[13]

[11] Krcmar (2010), S. 193.

[12] Synonym werden auch die Begriffe linear oder konzeptionell in der Literatur verwendet.

[13] Vgl. Henrich (2002), S. 34 ff.; vgl. Krcmar (2010), S. 193.

		Ablaufgestaltung	
		Sequentiell	**Iterativ**
Formalisierung	**Stark formalisiert**	Wasserfallmodell, V-Modell, V-Modell (200x), V-Modell XT, W-Modell, ...	Spiralmodell, RUP, Prototyping, OO Lifecycle-Modell, FDD, ...
	Wenig formal		Extreme Programming, Object Engineering Process, SCRUM, MDA, ...

Abb. 3.4 Überblick der Vorgehensmodelle zur Anwendungsentwicklung. (Entnommen aus: Krcmar (2010), S. 193)

Neben den sequentiellen Phasenkonzepten gibt es noch Vorgehensmodelle, die den iterativen Prozess der Anwendungsentwicklung widerspiegeln. Bei iterativen bzw. inkrementellen Vorgehen ist ein Rücksprung zu vorangegangenen Aktivitäten oder Phasen möglich. Krcmar definiert die Vorgehensmodelle nach dem Ablauf (sequentiell oder iterativ) und dem Grad der Formalisierung. Eine Einordnung gängiger Modelle ist in der Abb. 3.4 dargestellt[14]. Die Modelle sind nach ihrer ursprünglichen Konzeption eingeordnet, können unter anderen Aspekten aber auch in andere Quadranten eingeordnet werden. Beispielsweise lässt sich das Spiralmodell auch rein sequentiell betrachten, wenn pro Umlauf ein komplettes Phasenkonzept nach Wasserfallmodell durchlaufen wird.

Neben den bereits genannten Ansätzen zur Klassifikation von Vorgehensmodellen hat sich seit Anfang 2001 mit der Veröffentlichung des agilen Manifests eine weitere Unterscheidung etabliert: phasenorientiert versus agil. Cockburn beschreibt die agilen Gedankenmodelle als „... Verwendung von einfachen aber erfolgreichen Regeln zum Projektverhalten sowie der Gebrauch von mensch- und kommunikationsorientierten Regeln.“[15] Ziel der Anhänger von agilen Methoden ist es, schlanke (lean) Prozesse in der IT-Industrie einzuführen. Daher wird in der Literatur auch häufig von der „Entbürokratisierung der Softwareentwicklung und dem Verzicht einer dokumentengetriebenen Vorgehensweise“ gesprochen[16]. Auf Basis dieser Klassifikation herrscht in der Softwareentwicklung zunehmend eine Art „Glaubenskrieg“ zwischen den Anhängern der verschiedenen Modelle[17].

[14] Vgl. Krcmar (2010), S. 193 f.

[15] Cockburn (2003), S. 12.

[16] Schmietendorf (2013a), S. 14.

[17] Vgl. Ekssir (2013), S. 10.

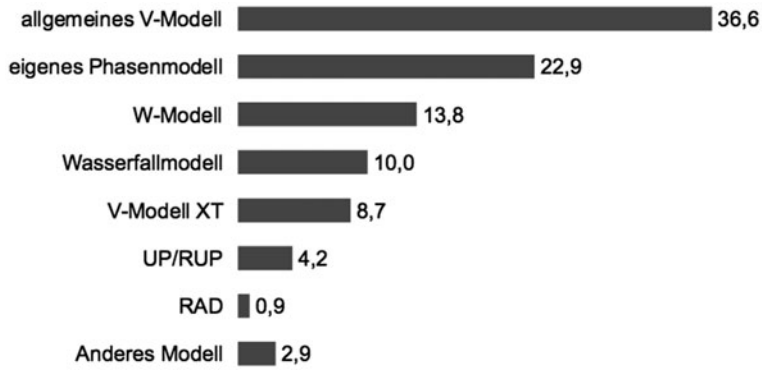

Abb. 3.5 Verbreitung phasenorientierter Modelle (Angaben in Prozent). (Entnommen von: HS Bremen et al. (2011))

Aufgabe eines hybriden Modells soll es sein, eine Brücke zwischen phasenorientierten und agilen Methoden zu schlagen. Aus diesem Grund werden die nachfolgenden Kapitel zur Vorstellung konkreter Modelle nach phasenorientierten und agilen Vorgehensmodellen gegliedert. In Kap. 3.4 werden die Grundideen der agilen Modelle vorgestellt und eine detaillierte Abgrenzung zu den vorher vorgestellten phasenorientierten Modellen vorgenommen.

3.3 Phasenorientierte Vorgehensmodelle

Phasenorientierte Modelle kennzeichnen sich durch eine Unterteilung der Systementwicklung in zeitliche Abschnitte. Das erste dieser Modelle wurde von Herbert D. Benington im Jahre 1956 und damit zeitgleich mit der Entwicklung der ersten Hochsprachen veröffentlicht. Mit seinem *stagewise model* schlug er eine phasenorientierte Vorgehensweise für die Softwareentwicklung vor, die bereits viele Elemente moderner Vorgehensmodelle in sich trug.[18] Durch die verschiedenen Entscheidungsstufen soll die Komplexität des kontinuierlichen Entscheidungsprozesses reduziert werden[19]. Aus diesem Grund werden die Phasen nach Zeitpunkten unterteilt, „. . . an denen Entscheidungen von grundsätzlicher Bedeutung zu fällen sind“[20]. Es gibt kein allgemein gültiges Phasenschema. Allerdings haben die im Folgenden vorgestellten Ansätze Ähnlichkeiten in ihrer Grobstruktur, bei der jede Phase sich durch typische Entscheidungen und Aufgaben charakterisieren lässt.

[18] Vgl. Hoffmann 2008, S. 493.

[19] Vgl. Hansen und Neumann (2005), S. 264.

[20] Ebd.

Im Folgenden werden das Wasserfallmodell und das V-Modell vorgestellt, die in Deutschland einen hohen Verbreitungsgrad haben.

3.3.1 Wasserfallmodell

Das Wasserfallmodell ist eine Weiterentwicklung des Softwarelebenszyklus-Modells und versucht, die stark idealisierten Annahmen zu beseitigen. Das Softwarelebenszyklus-Modell schreibt einen streng sequentiellen Entwicklungsprozess vor, der den Rücksprung zu einer vorhergehenden Phase nur unter Ausnahmen zulässt. Im Wasserfallmodell sind die einzelnen Phasen stärker untergliedert und die Wechselwirkungen zwischen den Phasen werden berücksichtigt.[21]

Das ursprüngliche Modell wurde von Royce im Jahr 1970 in dem Artikel „Managing the development of large Software Systems“ veröffentlicht. Royce selbst bezeichnete seine Vorgehensweise schlicht als „Implementation steps to develop a large program for delivery to a customer“. Er beschreibt lediglich den Ablauf der Phasen, nicht aber die genauen Arbeitsschritte, die in den einzelnen Phasen abgearbeitet werden,[22]

„Der heutige Name Wasserfallmodell wurde erst 11 Jahre später durch den amerikanischen Software-Ingenieur Barry W. Boehm geprägt.“[23] Der Name basiert darauf, dass „... die Ergebnisse einer Phase wie bei einem Wasserfall in die nächste Phase fallen.“[24]

Im Wasserfallmodell sind zum Softwarelebenszyklus-Modell folgende Erweiterungen zu finden[25]:

- Die Entwurfsphase ist in Grob-und Feinentwurf aufgeteilt. Zusätzlich wird eine Phase Installation eingeschoben.
- Jede Phase kann Rückkopplung zur vorhergehenden Phase liefern. Da die Kosten für Nacharbeiten über mehrere Phasen hoch sein können, beziehen sich die Rückkopplungen immer auf unmittelbar aufeinander folgende Phasen. Da das Wasserfallmodell von strikt sequentiellen Projektphasen ausgeht, sind Rücksprünge zur vorhergehenden Phase nur erlaubt, wenn die vorhergehenden Resultate fehlerhaft und zu korrigieren sind.
- Am Ende jeder Projektphase steht ein Validierungsprozess an. Die Validierung bezieht sich auf die Funktionalität des Informationssystems. Prototypen können bereits ab der Phase der Systemspezifikation für die Validierung eingesetzt werden. Damit ist über alle Phasen die Qualitätssicherung gewährleistet und nicht – wie im einfachen Vorgehensmodell – erst am Ende des Projekts.

---

[21] Vgl. Balzert (1998), S. 99; vgl. Hansen und Neumann (2005), S. 266 f.

[22] Vgl. Grechening et al. (2010), S. 374 f.; vgl. Hoffmann (2008), S. 493.

[23] Hoffmann (2008), S. 493.

[24] Balzert (1998), S. 99.

[25] Vgl. Hansen und Neumann (2005), S. 268 f.

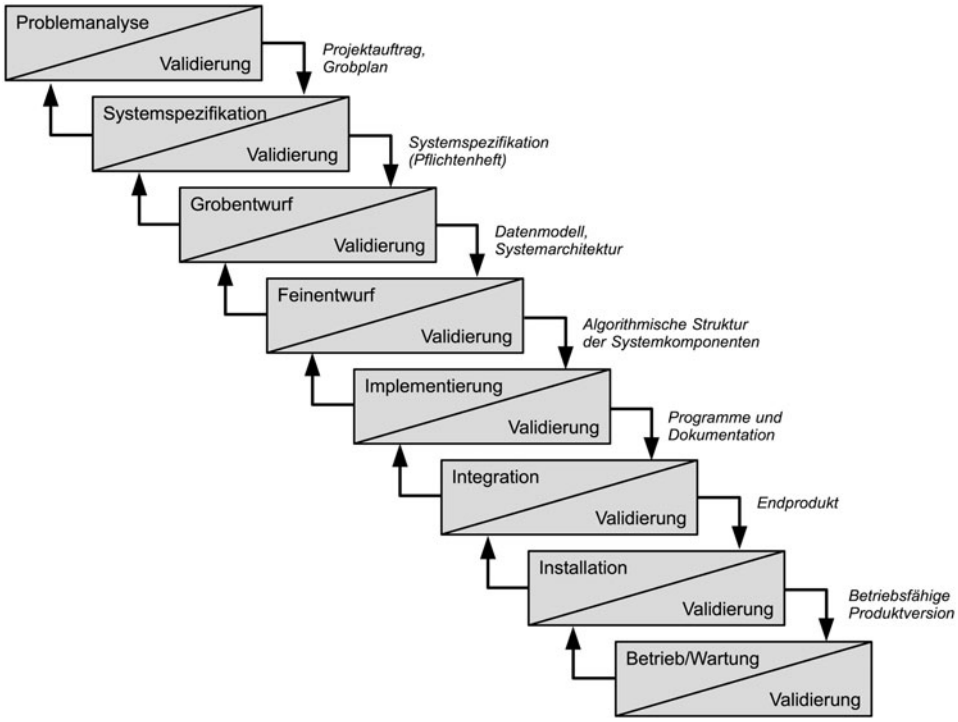

Abb. 3.6 Erweitertes Wasserfallmodell nach Boehm. (Entnommen aus: Hansen und Neumann (2005), S. 267 f.)

- Wird eine Phase erfolgreich abgeschlossen, so wird der Projektfortschritt in einem Dokument fixiert. In dem Modell von Royce werden damit sehr früh zwei Schlüsselkonzepte umgesetzt, die in den meisten modernen Vorgehensmodellen in ähnlicher Form vorhanden sind: Dokumente und Meilensteine.[26]

Abbildung 3.6 stellt den schematischen Ablauf des Wasserfallmodells nach Boehm dar: Wie in der Abbildung zu sehen, können die folgenden Phasen unterschieden werden[27]:

1. **Problemanalyse:** Es wird ein Konzept zur Prüfung der Machbarkeit für das zu entwickelnde Softwaresystem erstellt. In der Machbarkeitsstudie wird der Nutzen, die Umsetzbarkeit und die Vorteilhaftigkeit gegenüber alternativen Lösungsansätzen gesucht. Diese Phase kann nach einem evaluativen Modell weiter untergliedert werden. Bei einer erfolgreichen Validierung der Machbarkeit endet diese Phase in einem Projektauftrag und einer groben Planung.

[26] Hoffmann 2008, S. 494.

[27] Vgl. Hansen und Neumann (2005), S. 267 f.; vgl. Henrich (2002), S. 46.

2. **Systemspezifikation:** Der Auftragnehmer definiert seine Anforderungen in einem Pflichtenheft. Diese umfassen die erforderliche Funktionalität, die notwendigen Schnittstellen und die gewünschte Performance des Softwaresystems.
3. **Grobentwurf:** Mit dem Grobentwurf erfolgt eine grobe Spezifikation der Hard- und Softwarearchitektur, der Kontroll- und Datenstrukturen. Weiterhin werden erste Entwürfe der Benutzerhandbücher und Testpläne erstellt.
4. **Feinentwurf:** In dieser Phase erfolgt eine vollständige und verifizierte Spezifikation der Kontroll- und Datenstrukturen, der Schnittstellenbeziehungen und der Algorithmen. Es wird ein Übersichtsplan zur Implementierung der verschiedenen Programmkomponenten aufgestellt.
5. **Implementierung:** Anhand des Feinentwurfs erfolgt die Umsetzung des Softwaresystems.
6. **Integration:** Die einzelnen Programmkomponenten werden zu einem funktionsfähigen System zusammengesetzt.
7. **Installation:** Das System wird installiert und in Betrieb genommen. Zur Inbetriebnahme zählt die Übergabe der Benutzerhandbücher und die Schulung der Anwender.
8. **Betrieb/Wartung:** Innerhalb des Betriebs und der Wartung werden Fehlerkorrekturen vorgenommen.

3.3.2 V-Modell

Das V-Modell ist nach wie vor der Entwicklungsstandard für IT-Systeme des Bundes und beschreibt detailliert das Vorgehen bei der Entwicklung von IT-Systemen, die aus Hard- und/oder Softwarekomponenten bestehen können. Es handelt sich dabei um kein abstraktes Modell, sondern es ist ein in der Praxis vornehmlich für Großprojekte verbreitetes Vorgehenskonzept. Das V-Modell ist unterteilt in Submodelle, die sich mit den Aufgaben Systemerstellung, Qualitätssicherung, Konfigurations- und Projektmanagement beschäftigen.[28]. Der Schwerpunkt der Betrachtung des V-Modells in diesem Kapitel liegt bei der Systemerstellung mit ihren einzelnen Phasen (vgl. Abb. 3.7).

Die Abbildung macht deutlich, dass das „V“ eine mehrfache Bedeutung hat. Zum Einen steht es für das Vorgehen, zum Anderen spiegelt das Gesamtvorgehen die Form des Buchstaben „V“ wider. Dies wird erzeugt, indem die Phasen des Wasserfallmodells um qualitätssichernde Maßnahmen erweitert werden. Alle mit der Erstellung der Software verbundenen Aktivitäten befinden sich auf dem absteigenden Teil des „V“ und entsprechen in weiten Teilen den Phasen des Wasserfallmodells. Der aufsteigende Teil des „V“ wird durch Integrations- und Qualitätssicherungsaktivitäten gebildet.[29] Damit steht das „V“ zusätzlich für Verifikation und Validierung. In den Phasen zur Verifikation wird die Übereinstimmung des Systems mit der Spezifikation auf Korrektheit hin überprüft, während die Validierung

[28] Vgl. Hansen und Neumann (2005), S. 271; vgl. Henrich (2002), S. 54 f.

[29] Vgl. Henrich (2002), S. 57.

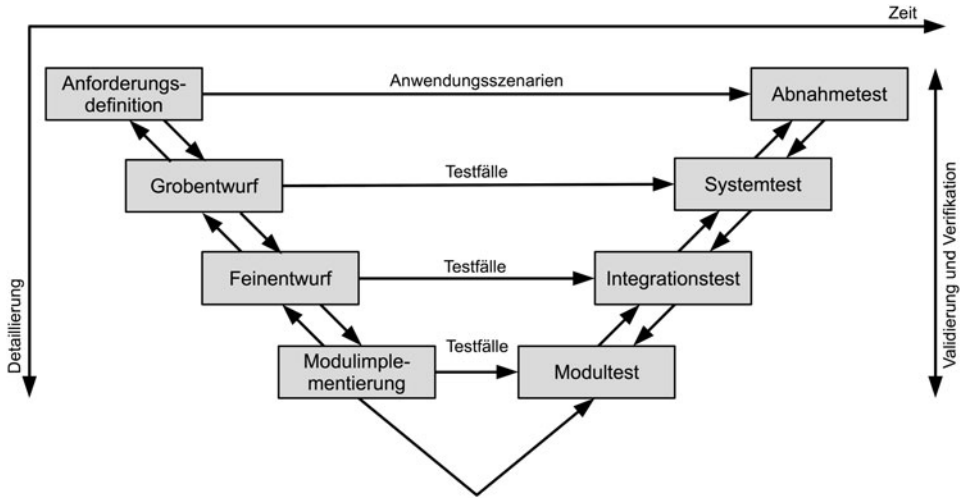

Abb. 3.7 Vereinfachte Darstellung des V-Modells. (Eigene Darstellung in Anlehnung an: Balzert (1998), S. 101)

die Angemessenheit des Systems in Bezug auf die Problemstellung sicherstellen soll. Dazu wird einer Phase auf der linken Seite eine qualitätssichernde Phase auf der rechten Seite gegenübergestellt[30].

Das V-Modell legt neben der Vorgehensweise sowohl die zu verwendenden Methoden als auch die funktionalen Eigenschaften der einzusetzenden Werkzeuge fest. Im V-Modell wird das Vorgehen in der Softwareentwicklung durch eine einheitliche und verbindliche Vorgabe von Aktivitäten und Ergebnissen strukturiert. Hinzu kommen noch die begleitenden Tätigkeiten. Jede Aktivität ist in Form einer Aktivitätenbeschreibung genau dokumentiert und erfüllt damit die Funktion einer Arbeitsanleitung. Damit trifft das V-Modell Regelungen, wie eine Aufgabe auszuführen ist[31]. Dies entspricht exakt der Grundidee der Vorgehensmodelle und der Beschreibung der Aspekte einer Aufgabe (vgl. Kap. 3, Abschn. 3.2).

Seit seiner Entwicklung 1992 wurde das V-Modell mehrfach überarbeitet und ist unter dem Namen V-Modell XT bekannt. Das XT steht für Extreme Tailoring und soll Anpassungen dieses umfassenden Regelwerks auch für kleinere Projekte erlauben. Die Art des Tailoring kann in ausschreibungsrelevantes und technisches Tailoring unterschieden werden. „Im ausschreibungsrelevanten Tailoring werden vor Projektbeginn die für das Projekt erforderlichen Aktivitäten und Produkte des V-Modells bestimmt. . . . Die so für das konkrete Projekt festgelegte Teilmenge. . . wird gemeinsam mit weiteren Vereinbarungen im Projekthandbuch festgeschrieben. Beim technischen Tailoring wird auf Basis der im Pro-

[30] Vgl. Balzert (1998), S. 103; vgl. Henrich (2002), S. 57; vgl. Krcmar (2010), S. 194.

[31] Vgl. Hansen und Neumann (2005), S. 272; vgl. Henrich (2002), S. 59.

jekthandbuch festgeschriebenen Ausführungsbedingungen entschieden, welche Aktivitäten durchzuführen sind."[32]

Anstelle von Submodellen nutzt das V-Modell in der XT-Variante aufeinander aufbauende Vorgehensbausteine. Der Vorgehensbaustein ist die zentrale Einheit des Tailoring. Er enthält alle Produkte, Aktivitäten und Rollen, die inhaltlich zusammen gehören.[33] Dadurch wird die projektspezifische Anpassung vereinfacht und das Vorgehen kann während des Projektablaufs situativ anhand der festgelegten Regeln angepasst werden.

Das V-Modell ist eines der umfassendsten Vorgehensmodelle, das durch Tailoring versucht, alle Projektgrößen abzubilden. Auf über 1000 Textseiten regelt das V-Modell die Detailschritte und die Koordination zwischen diesen durch Formulare. Durch die standardisierte Abwicklung von Projekten trägt es zu den Zielen bessere Planbarkeit, Deckelung der Entwicklungskosten, Qualitätssicherung und Verbesserung der Kommunikation bei. Allerdings ist der Umfang des V-Modells trotz Tailoring die große Schwachstelle des Vorgehensmodells. Dieser macht deutlich, dass es für große eingebettete Systeme optimiert ist[34]. Für Organisationen, die sich an Ausschreibungen im öffentlichen Dienst beteiligen, unterstützt das V-Modell diese mit Verweisen auf Vergaberegeln zur korrekten, rechtssicheren Ausschreibung und Vergabe. Zusätzlich ist das V-Modell kompatibel zu unterstützenden Prozessverfahren wie bspw. RUP, Capability Maturity Model Integration (CMMI) und ISO 9000[35].

3.4 Agile Vorgehensweisen

Die Vertreter und Vordenker der agilen Softwareentwicklung sehen ihre Sammlungen von Methoden und Best Practices nicht als Modelle, sondern eher als anpassbare Frameworks oder nur als Vorgehensweisen an. Um den Bezug zu den phasenorientierten Vorgehensmodellen herzustellen, wird im Verlauf der Begriff des Modells genutzt. Die Auswahl der in diesem Buch vorgestellten agilen Praktiken und Vorgehensweisen beruht auf einer Umfrage[36] des Unternehmens Version One unter mehr als 6000 Entwicklern im Jahr 2011. Abbildung 3.8 stellt die Ergebnisse in Form eines Kreisdiagramms im Uhrzeigersinn dar.

Nach den Ergebnissen der Umfrage nutzen die meisten Unternehmen Scrum oder Extreme Programming (XP). Um ein tieferes Verständnis für die agilen Methoden zu schaffen, werden anhand des agilen Manifests und den sieben Prinzipien des Lean Software Developments die wesentlichen Werte erläutert. Als spezielle Methoden werden die weit verbreiteten Methoden Scrum und XP detailliert vorgestellt (siehe Kap. 3, Abschn. 3.4.3 und 3.4.4).

[32] Henrich (2002), S. 60; vgl. auch Hansen und Neumann (2005), S. 274 f.

[33] Vgl. Höppner und Höhn (2005), S. 10 f.

[34] Vgl. Henrich (2002), S. 60.

[35] Vgl. Höppner und Höhn (2005), S. 10 f.

[36] Vgl. Version One (2011), S. 1, 4.

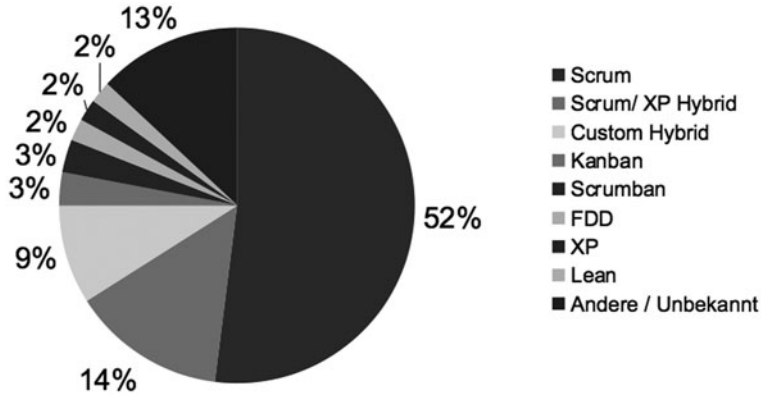

Abb. 3.8 Umfrageergebnis zum Einsatz agiler Methoden. (Verkürzt entnommen aus: Version One (2011), S. 4. (Darstellung im Uhrzeigersinn))

3.4.1 Das agile Manifest

Zu Beginn des Jahres 2001 fanden sich die Begründer verschiedener leichtgewichtigen Methoden aus der Softwareindustrie (darunter Schwaber und Sutherland [Scrum] und Beck [XP]) zusammen, um basierend auf ihren beruflichen Erfahrungen und Überzeugungen eine gemeinsame Basis herauszuarbeiten, die als Leitfaden für die agile Softwareentwicklung dienen sollte. Es entstand das sogenannte „Agile Manifesto". Abbildung 3.9 zeigt das agile Manifest in der deutschen Übersetzung.

Neben den Leitsätzen wurde auch der Begriff „agil" geprägt, da die Versammelten der Meinung waren, dass „leichtgewichtig" (lean) eher eine Abneigung gegen etwas vermittelt. Unterstützt werden die vier Leitsätze durch zwölf weitere Prinzipien die aus der *Lean Production* abgeleitet sind[37].

Das agile Manifest ist eine Reaktion auf die bürokratischen und formalen Prozesse der phasenorientierten Vorgehensmodelle. Zeitgleich werden agile Ansätze dahingehend kritisiert, chaotisch zu sein, schlechte Qualität und undisziplinierte Entwickler zu fördern[38].

3.4.2 Lean Software Development

Lean Software Development stellt die Anwendung der Lean Prinzipien auf die Softwareentwicklung dar.[39] „Bei Lean handelt es sich um eine Denkweise, die den Kunden und den eigentlichen Prozess der Wertschöpfung in den Mittelpunkt stellt: Wertschöpfung bezieht

[37] Vgl. Cockburn (2003), S. 282; vgl. Stober und Hansmann (2010), S. 36 f.

[38] Vgl. Appelo (2011), S. 20; vgl. Lami und Falcini (2009), S. 130.

[39] Vgl. Poppendieck (2003), S. xxiv.

Abb. 3.9 Das agile Manifest. (Quelle: http://agilemanifesto.org/iso/de/)

sich dabei auf all jenes, für das der Kunde zu zahlen bereit ist; alles Andere – die sogenannte Verschwendung – gilt es zu identifizieren und zu reduzieren.“[40] Ursprünglich stammt Lean aus der Automobil-Fertigung. Die ersten Benennungen gehen auf Taiichi Ohno und das Toyota Production System zurück.[41] Da sich die Softwareentwicklung jedoch erheblich von der Fertigung und der Logistik unterscheidet, ist die Überführung der Praktiken aus der Fertigung und dem Supply Chain Management nicht direkt übertragbar.[42]

Im Laufe der Jahre wurde das Lean-Gedankengut auf viele verschiedene Wirtschaftsaspekte ausgebreitet und es entstanden verschiedene Subdisziplinen wie z. B. Lean Management oder Lean Solutions. Für Mary und Tom Poppendieck ist die Softwareentwicklung eine Form der Produktentwicklung. Da Software meist Teil eines Produktes ist, kann die Softwareentwicklung auch als Untermenge der Produktentwicklung gesehen werden.[43]

Im Zentrum des Lean Software Developments stehen sieben Prinzipien. Diese wurden von Mary und Tom Poppendieck aufgestellt. Da andere Autoren hinsichtlich der Prinzipien auf die Literatur der Poppendiecks verweisen (siehe z. B. Schuh[44], Hibbs[45] und Appelo[46]), wird in diesem Buch zur Erläuterung der Prinzipien ausschließlich deren Literatur und deren englischen Originalbezeichnungen verwendet.

[40] Gundlach (2008), S. 294.
[41] Vgl. Womack (2013), S. 23.
[42] Vgl. Poppendieck (2007), S. 20.
[43] Vgl. ebd., S. 17.
[44] Vgl. Schuh (2005), S. 41.
[45] Vgl. Hibbs (2008), S. 16.
[46] Vgl. Appelo (2010), S. 25.

Für Mary und Tom Poppendieck sind Prinzipien fundamentale Wahrheiten, die sich im Laufe der Zeit oder in verschiedenen Umgebungen nicht ändern. Dadurch unterscheiden sie sich von Methoden, die sich abhängig von der Umgebung unterscheiden und im Laufe der Zeit an veränderte Rahmenbedingungen angepasst werden können. Bei Problemen in der Softwareentwicklung, die nicht genau einzelnen Methoden zugeordnet werden können, sollte man sich daher nach ihrer Ansicht auf Prinzipien konzentrieren.[47]

Da diese sieben Prinzipen die Werte, die hinter allen agilen Methoden stehen, veranschaulichen, werden sie im Folgenden erläutert:

- **Eliminate Waste:** Die Eliminierung von Waste (auf deutsch „Verschwendung“) ist das fundamentale lean Prinzip.[48] Die Definition von Verschwendung im Lean Software Development entspricht im wesentlichen der aus den allgemeinen Lean Prinzipien: „Waste is anything that does not add value to a product, value as perceived by the customer.“[49] Die Eliminierung von Verschwendung kann zu einer erheblichen Reduzierung der Durchlaufzeiten führen.[50]
- **Build Quality In:** Das Prinzip *Build Quality In* des Lean Software Developments besagt, dass Software eine hohe technische Qualität haben soll. Um dies zu erreichen, sollte die Software bereits bei der erstmaligen Erstellung nach jeder kleinen Anpassung getestet werden, so dass Fehler sofort nach ihrem Auftreten gefunden und behoben werden.[51] Außerdem sollte sichergestellt werden, dass die Qualität auch nach dem erstmaligen Erstellen der Software erhalten bleibt. Hierzu sind kontinuierlich Anpassungen an der Software notwendig.[52]
- **Create Knowledge:** Bei dem Wasserfallmodell, dem „Klassiker“ der Softwareentwicklungsprozesse, werden die verschiedenen Phasen streng sequentiell durchlaufen.[53] Nach der Auffassung von Mary und Tom Poppendieck lässt die strikte Abarbeitung eines Plans jedoch keinen Raum, um Wissen zu generieren, da die Möglichkeit verschiedene Wege auszuprobieren nicht gegeben ist.[54] Hierbei sollte eine Balance zwischen dem Experimentieren und dem Erstellen eines Plans gefunden werden. Ziel ist es, so viel Wissen wie möglich zu generieren und die Kosten dabei so gering wie möglich zu halten.[55]
- **Defer Commitment:** In der Regel sind zu Beginn eines Vorhabens nicht alle Anforderungen im Voraus bekannt oder der Kunde ist sich nicht bewusst, dass er etwas Bestimmtes möchte.[56] Um dieses Problem zu umgehen, empfehlen Mary und Tom

[47] Vgl. Poppendieck (2007), S. 19.
[48] Vgl. Poppendieck (2003), S. 2.
[49] Ebd., S. xxv.
[50] Vgl. Poppendieck (2007), S. 95
[51] Vgl. ebd., S. 25 ff.
[52] Vgl. ebd., S. 29.
[53] Vgl. Hoffmann (2008), S. 494.
[54] Vgl. Poppendieck (2003), S. 19.
[55] Vgl. ebd., S. 20.
[56] Vgl. Rupp (2013), S. 104.

Poppendieck „Concurrent Software Development“ („gleichzeitige“ Softwareentwicklung) einzuführen. Das bedeutet, dass das Design oder die Anforderungen zu Beginn noch nicht vollständig detailliert ausgearbeitet sein müssen, sondern mit einem abstrakten Ansatz gestartet werden kann und dieser während der Umsetzung Stück für Stück verbessert werden kann. Dies wird auch als iterative Entwicklung bezeichnet. Außerdem ist Concurrent Development der beste Weg, um mit sich ändernden Anforderungen umzugehen.[57]

- **Deliver Fast:** „Customers like rapid delivery. . . . For customers of software development, rapid delivery often translates to increased business flexibility.“[58] Daher sollte es möglich sein, die gewünschte Software schnell auszuliefern. Außerdem unterstützt das Prinzip „Deliver Fast“ das Prinzip „Defer Commitment“ (siehe oben). Mit einer schnellen Softwareentwicklung kann sich ein Unternehmen verschiedene Optionen vergleichsweise lange offen halten, bevor es sich entscheiden muss.[59]
- **Respect People:** „Respect people“ bedeutet, dass den Entwicklungsteams grobe Pläne und vernünftige Ziele gegeben werden und sie sich selbst organisieren können, um diese Ziele zu erreichen. Der Respekt entsteht dadurch, dass den Beteiligten nicht gesagt wird, was und wie sie es zu tun haben, sondern dass man sie ermutigt eigenständig Entscheidungen zu treffen.[60]
- **Optimize the whole:** Bei der Optimierung eines Software Entwicklungsprozesses sollte stets der komplette Prozess inkl. aller beteiligten Personen oder Parteien berücksichtigt werden. Optimierung an lediglich einem Teil des Prozesses, kann sogar zur Beeinträchtigung des gesamten Prozesses führen.[61] Dieses Prinzip gilt nicht nur innerhalb eines Unternehmens, sondern auch wenn verschiedene Unternehmen zusammen arbeiten.[62]

Die wesentlichen Inhalte dieser Prinzipien sind in beinahe allen agilen Methoden wie z. B. Extreme Programming und Scrum wiederzufinden

3.4.3 Extreme Programming

Extreme Programming (XP) ist ein agiles Methodenkonzept, das von Kent Beck definiert worden ist und auf gelebten Werten, Prinzipien[63] und Praktiken beruht. XP basiert auf dem Paradigma der Veränderung. Software ist kein statisches Produkt, sondern wird getrieben

[57] Vgl. Poppendieck (2003), S. 48 f.

[58] Ebd., S. 70.

[59] Vgl. Poppendieck (2003), S. 70.

[60] Vgl. Poppendieck (2007), S. 37.

[61] Vgl. ebd., S. 38 f.

[62] Vgl. Poppendieck (2003), S. xxvii.

[63] Becks Ausführungen zu den Prinzipien sind teilweise philosophischer Natur, sodass diese an dieser Stellenicht weiter betrachtet werden.

und verändert durch neue Anforderungen, Designänderungen, gesetzliche Regelungen u. v. m.[64]

Die Definition der Werte orientiert sich dabei stark am agilen Manifest und wird auf die fünf Schlagworte *Communication*, *Simplicity*, *Feedback*, *Courage* und *Respect* reduziert. Um die Geschwindigkeit der Softwareentwicklung zu erhöhen und trotzdem eine hohe Qualität zu gewährleisten, beschreibt Beck die folgenden Praktiken[65]:

- **Pair Programming:** Zwei Designer entwickeln gemeinsam an einem Arbeitsplatz eine Programmpassage. Während ein Entwickler codiert, schaut der andere zu und führt das Code-Review durch. Die Teams sollen in ihrer Besetzung regelmäßig wechseln, um auch das Wissen über das Design der Software an alle zu verbreiten.
- **Stories:** Stories sind in XP schriftlich fixierte Anforderungen des Kunden oder der Entwickler an neue Funktionalität. Das neue Feature wird entweder direkt durch den Kunden oder die Entwickler prosaisch formuliert und auf einer Karteikarte festgehalten.
- **Real Customer Involvement:** Der Kunde soll über den gesamten Entwicklungszyklus einer neuen Funktionalität einbezogen werden und den Entwicklern regelmäßig Feedback geben.
- **Cycles:** Die Entwicklung einer oder mehrerer neuer Funktionen wird in Zyklen oder Iterationen aufgeteilt. Dabei wird zwischen Entwicklungszyklen und Freigabezyklen unterschieden. Entwicklungszyklen können aus einem Zeitraum von einer bis drei Wochen bestehen, wobei der Freigabezyklus spätestens nach zwei bis fünf Iterationen stattfinden sollte. Jeder Entwicklungszyklus führt mindestens zu einem getesteten und funktionsfähigen Prototypen.
- **Continuous Integration:** Ein automatisierter, täglicher Build-Prozess der Software ist die Mindestvoraussetzung für Continuous Integration (CI). CI soll das Problem der Integration von Code in das allen Entwicklern gemeinsam zugängliche Tool für das Source-Code Management (SCM) lösen. Codeänderungen und die zugehörigen Unit-Tests werden täglich mehrmals durch die Entwickler in das SCM eingespielt und das CI-System prüft auf Veränderungen im SCM, baut die Software und führt die entsprechenden Tests durch (siehe auch Kap. 5, Abschn. 5.2.7).
- **Ten-Minute Build:** Der Ten-Minute Build ist die Bedingung für die Prozesslaufzeit bei CI. Der gesamte Prozess beim Bau eines Moduls bis zum getesteten Produktbaustein darf nicht länger als zehn Minuten dauern.
- **Test-First Programming:** Unter Test-First Programming oder auch Test Driven Development (TDD) wird das kontinuierliche Testen von einzelnen Modulen nach Änderungen verstanden. Da manuelles Testen zeit- und kostenintensiv ist, sollen die Testprozesse vollständig automatisiert werden (siehe Kap. 5, Abschn. 5.2.5).
- **Incremental Design:** Durch die Definition von Zyklen ist ein inkrementelles Design notwendig, wie es beim Prototyping genutzt wird.

[64] Vgl. Beck und Andres (2010), S. 11.

[65] Vgl. ebd., S. 37–69.

- **Shared Code:** Alle Entwickler sind gemeinsam verantwortlich für die Code-Basis. Das bedeutet, dass alle jederzeit Änderungen am Code vornehmen können.
- **Automatic Testing:** Nach Änderungen an einer Funktionalität sollten im Rahmen eines Regressionstests alle Testfälle durchgeführt werden. Die Durchführung manuell vorzunehmen ist zeitintensiv. Daher soll das System nach einer Änderung diese Tests automatisch durchlaufen und den Entwickler im Erfolgs- oder Fehlerfall informieren.
- **Daily Deployment:** Ziel des Daily Deployment oder auch Continuous Delivery (CD) ist es, täglich neue Funktionalität in Betrieb zu nehmen. Dadurch sollen besonders schwere Fehler zügig entdeckt werden bzw. sich nicht unbemerkt in das Produktivsystem einschleichen können.

Beck definiert für XP Zyklen. Diese sind allerdings nicht mit den Phasen eines Phasenmodells vergleichbar. Das liegt darin begründet, dass sich das Modell an Entwickler richtet und daher nur die Phase der Implementierung betrachtet. Diese lässt sich wiederum in Design, Test, Kodierung und Refaktorisierung unterteilen.

Cockburn definiert die Essenz von XP wie folgt: „XP ist eine hochdisziplinierte Methodik. XP verlangt die enge Befolgung der strikten Code- und Designstandards, strenge Verfahren für Unit-Tests, die jederzeit bestanden werden sollten, gute Akzeptanztests, konstantes paarweises Arbeiten, Gespür für einfaches Design und aggressive Refaktorisierung.“[66]

3.4.4 Scrum

Scrum ist ein agiles Framework, das von Schwaber und Sutherland kreiert worden ist. Die beiden Softwareentwickler haben Wert darauf gelegt, dass mit Scrum die vier Leitsätze aus dem agilen Manifest bestmöglich erreicht werden. Das Framework kennt kein ausgereiftes Phasenmodell, das ein Projekt über einen kompletten Lebenszyklus begleitet. Auch die Rolle des Projektmanagers gibt es bei Scrum nicht. Cohn beschreibt den Einsatz von Scrum für Projekte, in denen eine Lösung dringend und schnell erfolgen soll oder in denen die Anforderungen nicht vollständig definiert sind[67].

Das Framework besteht aus verschiedenen Regeln, die teilweise aus XP bekannt sind. Grundlage des rudimentären Phasenmodells, das Scrum zugrunde liegt, sind ein inkrementelles Design und die Entwicklung in Iterationen (siehe Abb. 3.10). Ein Scrum-Projekt startet mit der Idee zur Entwicklung eines Software-Systems. Marktbedingungen oder Systembedingungen können zum Zeitpunkt der Entstehung noch nicht festgeschrieben bzw. verbindlich sein, sodass das Konzept zu Anfang noch nicht vollständig ausgereift sein muss. Der Kunde (Product Owner) stellt einen Plan seiner gewünschten funktionalen und nicht-funktionalen Anforderungen auf und schreibt diese in einem Product Backlog fest. Innerhalb des Product Backlogs werden die Anforderungen in Zusammenarbeit mit den

[66] Cockburn (2003), S. 225.

[67] Vgl. Schwaber (2004), S. ix.

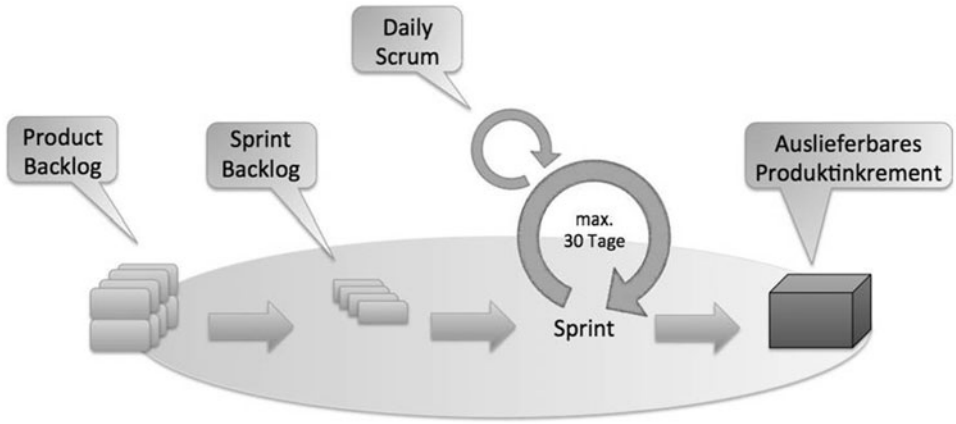

Abb. 3.10 Scrum im Überblick. (Entnommen aus: Pichler (2008), S. 7)

Entwicklern (Team) priorisiert[68]. Diese Art des Vorgehens ist mit den XP-Prinzipien *Real Customer Involvement* und *User Stories* vergleichbar.

Die Priorisierung in Zusammenarbeit mit den Entwicklern ist notwendig, denn diese entscheiden, welche Funktionalitäten innerhalb einer Iteration implementiert werden können. Diese priorisierten Anforderungen werden im Sprint Backlog erfasst. Eine Iteration in Scrum heißt Sprint und dauert maximal 30 Tage. Am Ende der Iteration sollte ein Prototyp der Software vorliegen, der ausgewählte Funktionalitäten aus dem Product Backlog demonstriert. Jede einzelne Funktionalität kann sich einen oder mehrere Sprints lang in der Entwicklung befinden.[69]

Im Gegensatz zu XP gibt Scrum keine Methoden zur Softwareentwicklung vor.[70] Jedoch gibt es jeden Tag eine 15-minütige Besprechung im Stehen (Daily Scrum), in der jeder Entwickler einen Fortschrittsbericht abliefert, über die nächsten Schritte unterrichtet und benennt, welche auftretenden Probleme er zu bewältigen plant.

3.5 Kritische Bewertung der Modelle

Neben den Eigenschaften wie Strukturierung in Phasen, Skalierbarkeit, Flexibilität und Praxisnähe der genannten Modelle gehören auch praktische Merkmale, die sich aus dem in Abb. 3.11 dargestellten Software Engineering Strukturbaum ableiten lassen.

[68] Vgl. Schwaber (2004), S. 8.

[69] Vgl. ebd., S. 10–14.

[70] Häufig werden aus XP bekannte Methoden wie z. B. *Pair Programming* auch in Scrum genutzt.

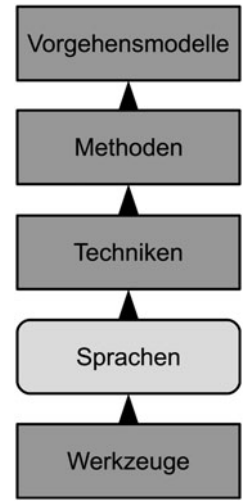

Abb. 3.11 Software Engineering-Strukturbaum. (Eigene Darstellung in Anlehnung an: Sneed (2007a), S. 56)

Vorgehensmodelle bilden den Kern des Software Engineerings. Methoden sind Verfeinerungen einzelner Tätigkeitsbereiche und stellen abstrakte Anleitungen zur Lösung eines bestimmten Problems dar. Eine Methode (z. B. objektorientierte Entwurfsmethode) wird wiederum durch eine Technik wie Unified Modelling Language (UML) praktisch umsetzbar. Softwareingenieure wenden diese Techniken vorzugsweise mit Werkzeuge an, die ihnen einen Teil der Arbeit abnehmen und klare Richtlinien vorgeben. Hierbei ist zu beachten, dass es für verschiedene Klassen von Programmiersprachen (funktional, imperativ, objektorientiert) unterschiedliche Werkzeuge existieren. Methoden, Techniken und Werkzeuge sollten auf das Vorgehensmodell und die genutzte Programmiersprache abgestimmt sein.[71]

Auf die Techniken wird bei der Bewertung nicht weiter eingegangen werden, da sich einige Techniken etabliert haben, die unabhängig vom Modell oder der verwendeten Programmiersprache genutzt werden können. Ein weiterer Schwerpunkt der Bewertung wird auf unterstützende Prozesse wie Risikomanagement und Qualitätssicherung gesetzt.

Die folgenden Kapitel beschreiben die Bewertungskriterien und prüfen, ob ein Vorgehensmodell die Kriterien erfüllt. In der abschließenden Zusammenfassung wird die Erfüllung bezogen auf die einzelnen Vorgehensmodelle tabellarisch dargestellt.

3.5.1 Struktur des Vorgehens

Ist hinter einem Vorgehensmodell ein Phasenmodell hinterlegt, so gibt dies einem Projekt einen ordnenden Rahmen. Dadurch lässt sich ein Vorgehensmodell für Projekte mit einem festen Endzeitpunkt, einem definierten Funktionsumfang und einem vorgegebenen Budget realisieren. Die Praxis hat gezeigt, dass Modelle wie das V-Modell oder der RUP für große Projekte und eingebettete Systeme gut geeignet sind.

[71] Vgl. Sneed (2007a), S. 55 f.

Ein agiles Vorgehen wie XP oder Scrum konzentriert sich auf den reinen Prozess der Softwareerstellung. Das Vorgehen ist iterativ. Agile Projekte haben einen Anfang, aber kein definiertes Ende. Die Vertragsgestaltung ist daher problematisch, da zu einem vorgegebenen Ende nur ein variabler Leistungsumfang zugesichert werden kann, denn das Design des Endprodukts entsteht während der Entwicklung. In XP werden alle administrativen Aufgaben auf den Kunden übertragen[72].

Für Festpreisprojekte sind alle vorgestellten Vorgehensmodelle wenig geeignet, denn die Aufwands- und Kostenschätzung findet in den Modellen zu wenig Berücksichtigung. Obwohl das V-Modell XT für den öffentlichen Dienst konzipiert wurde und 2/3 aller Projekte in Form eines Werkvertrags vergeben werden, findet sich in den Phasenmodellen keine Phase, in der die Aufwandsschätzung explizit eingeordnet ist. Für die Aufwandsschätzung wäre ein ausführliches Pflichtenheft erforderlich. Der Vertrag wird aber bereits vor Erstellung des Pflichtenhefts zwischen Auftragnehmer und Auftraggeber geschlossen. Demzufolge basieren Aufwands- und Zeitkalkulation ausschließlich auf Informationen des Lastenhefts[73].

Mögliche Lösungsansätze nach Sneed sind eine Verschiebung des Vertragsabschlusses bis zur Fertigstellung des Pflichtenheftes, gestaffelte Verträge, eine Erweiterung des Lastenheftes oder die Erfindung einer neuen Schätzmethode. Die praktikabelste Lösung scheint hierbei die Erweiterung des Lastenheftes um die Erweiterung der Objekte, Schnittstellen und Anwendungsfälle[74].

3.5.2 Praxisnähe

Die Akzeptanz und Praxisnähe eines Vorgehensmodells zeigt sich in der Häufigkeit seiner Nutzung. Je häufiger ein Modell in Projekten angewendet wird, desto praxisnäher ist es. Ein weiterer Aspekt ist die Form des Modells. Einige der in der Literatur genannten Modelle sind sehr abstrakt und daher für die Praxis nur bedingt geeignet. Daher werden im Folgenden die Modelle auf Nutzungsgrad und Abstraktheit geprüft.

Gemäß der Studie von Version One und der Software-Umfrage 2011 sind die vorgestellten Vorgehensmodelle Wasserfallmodell, V-Modell, Scrum und XP in der Praxis akzeptiert, denn sie bieten den Projektverantwortlichen wesentliche Organisations- und Arbeitshilfen[75]. Allerdings ergibt sich aus der Kritik auch, dass die vorgestellten Modelle für Festpreisprojekte nur bedingt geeignet sind und dementsprechend anzupassen sind. Rein agile Ansätze nach Scrum oder XP eignen sich für Projekte mit einem festen Budget und festen Liefertermin wenig. Sie könnten insofern angepasst werden, als dass sie dem Product Owner Schranken bezüglich seiner Anforderungen aufzeigen. Auch lässt sich im

[72] Vgl. Stephens und Rosenberg (2003), S. 81, 133.

[73] Vgl. Sneed (2007a), S. 59, 61 f.

[74] Vgl. ebd., S. 63.

[75] Vgl. Kütz (2009), S. 9.

Vorfeld keine exakte Kosten- und Nutzenanalyse erstellen, sofern nicht bereits vergleichbare Projekte erfolgreich durchgeführt worden sind.

Für das Wasserfallmodell und das V-Modell gelten die Aussagen aus Kap. 3, Abschn. 3.5.1. Für Festpreisprojekte sind diese Modelle geeignet, sofern die Vertragsgestaltung eine nachträgliche Anpassung erlaubt oder das Lastenheft um zusätzliche Informationen erweitert wird. Weitere Maßnahmen zur Optimierung wurden in Kap. 3, Abschn. 3.5.1 erläutert.

3.5.3 Skalierbarkeit

Methoden der Vorgehensmodelle skalieren unter verschiedenen Aspekten. Dazu gehören[76]:

- Produktarchitektur
- Entwicklungsprozess
- Unternehmensorganisation
- Teamgröße
- Abzuliefernde Funktionalität zu einem festen Zeitpunkt

Ursprünglich wurden agile Ansätze wie Scrum oder XP für jede Projektgröße konzipiert. Stephens und Rosenberg haben allerdings anhand von empirischen Untersuchungen bewiesen, dass sich agile Ansätze für Großprojekte kaum eignen. Andererseits können phasenorientierte Modelle, wie das V-Modell, trotz der Möglichkeit des Tailoring in kleinen Projekten nur unter großem zusätzlichen Aufwand eingesetzt werden, da allein das Modell schon auf über mehr als 1000 Seiten dokumentiert ist und somit deutlich macht, dass es ursprünglich für große eingebettete Systeme entwickelt wurde.[77]

Das V-Modell bspw. dokumentiert alle notwendigen Prozesse und Methoden sehr umfangreich und vollständig, erhebt aber den Anspruch, dass je nach Projektgröße nicht alles anzuwenden ist. Allerdings ist die Anpassung und das Einlesen in die umfangreichen Beschreibungen sehr aufwändig. Daher gelten diese Modelle auch als bürokratisch und unflexibel. Aus diesem Grund wurden agile Ansätze wie Agile Unified Process (AUP) oder Open Unified Process (OUP) entwickelt.[78]

Die Empfehlung von Stephens und Rosenberg sowie Henrich lautet, dass die Entscheidung für ein Vorgehensmodell anhand der Projektgröße getroffen werden sollte[79]. Schwaber zeigt am Beispiel von einem 800 Mann starken Projekt, dass dieses in 100 kleine Teams mit jeweils acht Mitgliedern heruntergebrochen werden kann. Die Koordination der 100 Teams ist dann allerdings nur unter erheblichem Aufwand und der Schaffung klarer Schnittstellen

[76] Vgl. Stephens und Rosenberg (2003), S. 314.

[77] Vgl. Henrich (2002), S. 54; vgl. Stephens und Rosenberg (2003), S. 300.

[78] Vgl. Appelo (2011), S. 27.

[79] Vgl. Stephens und Rosenberg (2003), S. 23.

möglich. Andernfalls blockieren sich die Teams gegenseitig und die Entwicklung verzögert sich.[80]

An phasenorientierten Modellen wird häufig kritisiert, dass diese sich nur schlecht skalieren und anpassen lassen. Aber auch Scrum oder XP erfordern die Berücksichtigung aller vorgeschriebenen Methoden und Prinzipien oder sogar die Umstellung der Unternehmenskultur und – organisation.[81] Auf die Auswirkungen auf die Unternehmenskultur wird in Kap. 3, Abschn. 3.5.5 genauer eingegangen.

3.5.4 Vorgehen und Flexibilität

Die Flexibilität hängt vom Vorgehen (sequentiell oder iterativ) ab. Die Reaktion auf Veränderung erfolgt kurzfristig und kann erfolgreich in das Produkt bzw. einen Prototypen übernommen werden. Beck postuliert nicht die Veränderung als Problem, sondern den Umgang mit der Veränderung. Starre, phasenorientierte Modelle können mit kurzfristigen Veränderungen weniger gut umgehen, als dies mit agilen Paradigmen möglich ist. Hinzu kommen wirtschaftliche Aspekte, wie der aktuelle und der zukünftige Wert eines Softwareproduktes. Je eher ein Produkt zur Verfügung steht, umso vorteilhafter ist dies für die Organisation.[82]

Phasenorientierte Modelle gehen davon aus, dass bereits frühzeitig ein sorgfältig aufgestellter, langfristiger Plan vorliegt, der zukünftiges Design und zukünftige Anforderungen berücksichtigt. Agile Methoden schaffen den Wert früher. Langfristige Planung nach phasenorientiertem Design liefert den Wert später zurück und bewirkt, dass keine Fehler für die Zukunft gemacht werden[83]. Dieses in der agilen Entwicklung als *emergent design* bezeichnete Vorgehen, kann durch eine schlechte Architektur im Vorfeld zu hohem Refactoring-Aufwand führen und somit zu nachträglichen hohen Kosten[84].

Die phasenorientierten Vorgehensmodelle leiden nicht an mangelnder Flexibilität, sondern eher an der Art und Weise des Vorgehens. Durch den sequentiellen Charakter ist es nicht möglich, kurzfristig auf Änderungen zu reagieren. Sequentielle Modelle funktionieren dann sehr gut, wenn die Stakeholder formalisierte Anforderungen auf einer halbwegs konstanten und strukturierten Basis anbieten können. „Ein derart einheitlicher Informationsfluss wird zur verlässlichen Größe und führt auch zu ziemlich genau planbaren Zeiträumen im Entwicklungsprozess.“[85] Spielen besonders nichtfunktionale Anforderungen wie Performance, Sicherheit und Zuverlässigkeit oder restriktive Standards eine Rolle, so bietet sich das sequentielle Vorgehen mit formalisierten Dokumentationen an, um den üblichen Standards verlässlich zu genügen[86].

---

[80] Vgl. Schwaber (2004), S. 119–132.

[81] Vgl. Stephens und Rosenberg (2003), S. 82.

[82] Vgl. Beck und Andres (2010), S. 25.

[83] Vgl. Stober und Hansmann (2010), S. 97 f.

[84] Vgl. Stephens und Rosenberg (2003), S. 295.

[85] Dechko (2012).

[86] Vgl. ebd.

3.5.5 Unternehmenskultur

Der Begriff Unternehmenskultur (eigentlich Organisationskultur) stammt aus der Organisationstheorie und beschreibt die Entstehung und Entwicklung kultureller Wertemuster. Demnach bildet sich in einem Unternehmen eine spezifische Kultur heraus, die das Verhalten von Individuen in Organisationen maßgeblich bestimmt. Sie ergibt sich aus dem Zusammenspiel von Werten, Normen, Denkhaltungen und Paradigmen. Die Kultur prägt das Zusammenleben in der Organisation sowie die Außenwahrnehmung.

Linssen hat 2010 im Aufsatz *Agile Vorgehensmodelle aus betriebswirtschaftlicher Sicht* die Auswirkungen agiler Ansätze auf die Unternehmenskultur und die Gesamtorganisation untersucht. Untersucht wurden die Aspekte Arbeitsteilung, Spezialisierung, Koordination, Konfiguration, Formalisierung und Delegation mit den folgenden Ergebnissen[87]:

- **Arbeitsteilung:** In agilen Ansätzen wird nicht explizit zwischen verschiedenen Rollen differenziert. Es gibt folglich keine funktionale Stellenbildung. Die Entwickler bearbeiten die ihnen übertragenen Aufgaben deutlich umfassender. Die Stellenbildung ist demnach eher prozessorientiert und hat weitreichende Konsequenzen auf die Qualifikation der einzelnen Mitarbeiter. Der Bedarf an umfassendem und gut qualifiziertem Personal ist hoch. Auf Teamebene soll alles Spezialwissen der sonst in einem Projekt vorhandenen Rollen gebündelt sein.
- **Spezialisierung:** Im agilen Ansatz sind die horizontale und die vertikale Spezialisierung schwach ausgeprägt. Unter horizontaler Spezialisierung wird ausgedrückt, welchen Aufgabenumfang eine Person wahrnimmt. Bei der vertikalen Spezialisierung findet eine qualitative Trennung zwischen Durchführungsaufgaben und den Kontrollaufgaben statt. Die schwache Ausprägung der Spezialisierung bei den Teammitgliedern erhöht ebenfalls die Anforderung an die Qualifikation. Selbstmanagement wird zur essentiellen Qualifikation.
- **Koordination:** Koordination ist notwendig, um die durch Arbeitsteilung entstandenen Teile auf ein gemeinsames Ziel abzustimmen. Die Koordination durch Selbstabstimmung wird in agilen Gruppen favorisiert. Die persönliche Anweisung (ein Vorgesetzter gibt einem Untergebenen Anweisungen) findet sich in agilen Ansätzen kaum.
- **Konfiguration:** Unter Konfiguration wird das Leitungs- oder Stellensystem verstanden. Da in agilen Teams keine Vorgesetzten die Richtung vorgeben, existiert auch keine Gliederungstiefe oder Leitungsspanne.
- **Formalisierung:** Unter Formalisierung werden schriftlich fixierte Regeln in Form von Schaubildern, Stellenbeschreibungen, Prozessdiagrammen, Leistungsdokumentationen, etc. verstanden. Schriftlich fixierte Regeln sind faktisch kein Bestandteil agiler Ansätze.

[87] Vgl. Linssen (2010), S. 46–51.

- **Delegation:** Delegation beschreibt, wer innerhalb einer Organisationsstruktur Entscheidungen treffen darf. Die Entscheidungskompetenz obliegt dem Team, denn es steht in direktem Kontakt mit dem Kunden.

Diese Ausführungen zeigen, dass der reine Einsatz von agilen Ansätzen in Unternehmen erhebliche organisatorische Veränderungen erfordert. Funktionale Arbeitsteilung und Trennung zwischen Durchführungs- und Entscheidungsaufgaben werden reduziert. Durch den Abbau von Ein- oder Mehrliniensystemen wird die organisatorische Komplexität reduziert. Die zu erfüllenden Aufgaben sind komplexer und umfangreicher und erfordern entsprechend qualifizierte Mitarbeiter. Neben einer guten fachlichen Qualifikation benötigen die Mitarbeiter auch hervorragende Soft-Skills, insbesondere in den Bereichen Selbstmanagement, Konfliktmanagement und Kommunikation. Die Koordination durch persönliche Weisung wird durch Selbstbestimmung abgelöst[88].

3.5.6 Methoden

Methoden bezeichnen in der Softwareentwicklung ein systematisches Vorgehen. Sie sind für ein Vorgehensmodell notwendig, um die Basis für Techniken und Werkzeuge zu bilden. Allerdings sind Methoden hinsichtlich ihrer Praktikabilität zu prüfen, wie z. B. das Prinzip Shared Code bei XP: Auf der einen Seite ist jeder verantwortlich für die Codebasis. Auf der anderen Seite gibt es keine fest geordneten Verantwortlichkeiten. In großen Projekten kann so ein Prinzip zu Chaos und einer instabilen Codebasis führen[89].

Von diesen Methoden gibt es in der agilen Softwareentwicklung noch weitere. Stephens und Rosenberg[90] haben diese im Projektalltag untersucht und halten nur wenige für den Praxiseinsatz tauglich. Nur die als praxistauglich eingestuften Methoden bilden die Basis für das hybride Modell und werden in Kap. 5, Abschn. 5.2 detailliert vorgestellt.

Das phasenorientierte V-Modell verfügt über eine Methodenzuordnung. Es wird beschrieben, welche Aktivitäten mit welchen Methoden durchzuführen und wie die Ergebnisse darzustellen sind. An diese Methodenzuordnung ist auch eine Werkzeugzuordnung geknüpft, die beschreibt, welche funktionalen Anforderungen die Werkzeuge zu erfüllen haben. Welche Auswirkungen diese Einschränkungen auf die Kreativität haben können, ist im nachfolgenden Kapitel beschrieben.

Die für ein Festpreisprojekt notwendigen Methoden, wie Kosten- und Nutzenanalyse sowie das ausführliche Requirements Engineering im Vorfeld einer Angebotsabgabe, sind in den Modellen Scrum und XP nicht beschrieben. Das V-Modell beruft sich bei der Kosten- und Nutzenanalyse auf die klassischen Projektmanagementmethoden.

---

[88] Vgl. Linssen (2010), S. 51.

[89] Vgl. Stephens und Rosenberg (2003), S. 63.

[90] Vgl. ebd.

Es kann also festgehalten werden, dass weder agile noch phasenorientierte Vorgehensweisen einem industriellen Softwareentwickler einen vollständigen Methodenansatz für Festpreisprojekte mitgeben.

3.5.7 Techniken und Werkzeuge

Im gewerblichen Einsatz benötigen Projektbeteiligte Werkzeuge, die sie bei der Anwendung von praktizierten Methoden und Techniken unterstützen und sie schrittweise durch den Prozess leiten. Diese Steuerung wird durch Werkzeuge übernommen. Bei Werkzeugen handelt es sich um fest codierte Techniken, die die Arbeit erleichtern und den Benutzer in eine bestimmte Bahn lenken. Werkzeuge sind einerseits nützlich, aber schränken andererseits den Benutzer in seiner gestalterischen Freiheit ein. Kreative Menschen werden somit in ihrem Wirken beschnitten.[91]

Es ist nicht praktikabel, bestimmte Werkzeuge für ein Vorgehen festzuschreiben, wenn diese nicht unbedingt durch gesetzliche oder regulatorische Anforderungen eines Projekts notwendig sind. Das V-Modell lässt sich ohne geeignete CASE-Unterstützung allerdings kaum handhaben[92]. Werkzeuge sollten für das jeweilige Projekt individuell ausgewählt werden, wenn sie in Abhängigkeit zu einer bestimmten Sprache stehen. Es gibt aber auch Werkzeuge, deren Verwendung in jedem modernen Projekt zu empfehlen ist. Dazu gehört neben einem Daily Build (siehe Kap. 3, Abschn. 3.4.3) auch ein Defect Reporting Tool. Dieses wird dazu genutzt, Defects in einer Software zu erfassen und zu dokumentieren. Häufig werden diese Werkzeuge in agilen Projekten auch für die Dokumentation neuer Anforderungen herangezogen.

3.5.8 Unterstützende Prozesse

Der Unterstützungsprozess bezeichnet in der Betriebswirtschaftslehre betriebliche Prozesse, die einen oder mehrere Kernprozesse unterstützen. Wichtigstes Merkmal von unterstützenden Prozessen ist, dass sie nur mittelbar zur Wertschöpfung beitragen und aus der Organisation herausgetrennt und an Dritte ausgelagert werden können.

Das V-Modell beschreibt drei unterstützende Prozesse, die die eigentliche Systemerstellung begleiten. Das sind die Qualitätssicherung, das Konfigurationsmanagement und das Projektmanagement. Alle diese Prozesse lassen sich an Externe vergeben, die die dort definierten Aufgaben übernehmen. So ließen sich die Systemerstellung von einem Unternehmen A, das Projektmanagement von einem Unternehmen B oder dem Auftragnehmer, die Qualitätssicherung von einem darauf spezialisierten Unternehmen C und das Konfigurationsmanagement von einem Dienstleister D durchführen. Diese Auslagerung kann

[91] Vgl. Sneed (2007a), S. 56.

[92] Vgl. Balzert (1998), S. 113.

jedoch die Komplexität von Softwareprojekten erhöhen, da z. B. zusätzliche Kommunikationsschnittstellen erforderlich sind, an denen Informationsverluste entstehen können. Diese wiederum bewirken, dass sich aufgrund von Abstimmungsschwierigkeiten der Zeitaufwand für ein Projekt erhöht.

Da agile Ansätze die enge Zusammenarbeit zwischen Auftraggeber und Auftragnehmer präferieren, könnten hier Aufgaben aus den oben genannten Prozessen durch Externe übernommen werden. Diese werden allerdings so stark in den Prozess der Systemerstellung eingebunden, dass eine Unterscheidung der Prozesse nicht möglich ist. Das Konfigurationsmanagement wird von einem agilen Team ebenso übernommen, wie das Projektmanagement durch den Scrum Master oder Product Owner. Die Qualitätssicherung ist integraler Bestandteil der agilen Softwareentwicklung. Aufgrund der Wichtigkeit einer Qualitätssicherung für den Projekterfolg, wird dies im Kap. 3, Abschn. 3.5.8.2 weiter ausgeführt.

Zusammenfassend lässt sich festhalten, dass agile und phasenorientierte Vorgehen die gleichen unterstützenden Prozesse einsetzen und sich lediglich in ihrer organisatorischen Umsetzung geringfügig unterscheiden.

3.5.8.1 Risikomanagement

Unter einem Risiko wird die Abweichung von einem gesteckten Ziel verstanden. Das Risikomanagement beschreibt den planvollen Umgang mit Risiken.[93]

Es lassen sich drei Kerntechniken innerhalb der Phasen des Risikomanagements identifizieren[94]:

1. **Risikoanalyse:** Die Risikoanalyse wird zur Identifikation und Bewertung von Risiken eingesetzt und dient dazu, die Unsicherheit über potenzielle Gefahren einzudämmen. Es ist die wichtigste Stufe im Risikomanagement und eine kontinuierliche Aufgabe, die potenziellen Risiken zu erfassen und zu überwachen.
2. **Risikomatrix:** Mit der Risikomatrix werden Unsicherheiten über die Messung der Risiken reduziert. Die Risikofaktoren werden in einer Matrix erfasst und deren Eintrittswahrscheinlichkeit und das vermeintliche Schadenausmaß zugewiesen. Anhand der ermittelten Risikomatrix lässt sich das Gesamtrisiko für ein Projekt ermitteln.
3. **Risikobewertung:** Als Basis für die Risikobewertung dient die Risikomatrix. Es wird überprüft, welche Risiken zu vermeiden sind (Risikovermeidung) und bei welchen Risiken der Eintritt akzeptiert wird (Risikoakzeptanz). In der Risikobewertung wird ebenfalls überprüft, wie Risiken untereinander zusammenhängen und welche Maßnahmen zur Risikovermeidung, Risikoakzeptanz, Risikominderung und Risikobegrenzung ergriffen werden.

[93] Vgl. Krcmar (2010), S. 570 f.

[94] Vgl. Coyle und Conboy (2009), S. 143 f.

Agile Methoden dienen u. a. dazu, bekannte Risiken in Softwareprojekten zu reduzieren (Risikominderung). Dazu werden phasenorientierte Risikomanagement-Techniken, die längere Betrachtungszeiträume der Risikoanalyse nutzen, durch agile Ansätze auf wenige Wochen (Länge eines Sprints) reduziert werden. Eine Studie von Coyle und Conboy zum Risikomanagement mit DSDM[95] hat ergeben, dass sich das Risikomanagement weder in agilen noch in phasenorientierten Projekten beschneiden lässt; alle Stufen sind erforderlich[96].

Im Umkehrschluss bedeutet dies, dass in jedem Softwareprojekt ein Prozess zum Risikomanagement verankert sein sollte. Phasenorientierte und agile Vorgehen, die dies nicht in ihren Modellen vorsehen, können sich an dem klassischen Prozess des Risikomanagements orientieren und diesen parallel zum eigentlichen Vorgehen etablieren. Bei phasenorientierten Vorgehen lässt sich die Risikoanalyse nach mindestens dem Abschluss einer Phase wiederholen und bei agilen Vorgehen lässt sich nach jeder Iteration eine Risikoanalyse durchführen.

3.5.8.2 Qualitätssicherung

Die Qualitätssicherung ist Bestandteil eines Projekts und wird in interne und externe Qualitätssicherung unterschieden. Die externe Qualitätssicherung wird von nicht in das Projekt eingebundene Experten durchgeführt. Bei der internen Qualitätssicherung sind Projektmitarbeiter direkt für die Qualität ihrer Arbeitsergebnisse verantwortlich. Externe und interne Qualitätssicherung sind bereits Bestandteil der Projektplanung. Somit kann der Zeitaufwand für diese Tätigkeiten frühzeitig berücksichtigt werden.[97]

Mangelnde Bereitschaft zur Qualitätssicherung führt häufig zu einer halbherzigen Umsetzung des Qualitätsmanagements. Die Hauptkritikpunkte sind dabei[98]:

- Klassische Verfahren zum Qualitätsmanagement sind stark prozessorientiert und stellen allein keine Produktqualität sicher.
- Prozess-Standardisierung orientiert sich häufig an den größten und anspruchsvollsten Szenarien und ist für kleinere und mittlere Softwareprojekte überdimensioniert.
- Um eine externe Auditierbarkeit zu gewährleisten, ist häufig die Erstellung von Dokumenten vorgesehen, die das eigentliche Projektteam für sich als wenig hilfreich und als bürokratische Zusatzarbeit betrachtet.

---

[95] Die Dynamic Systems Delivery Method (DSDM) gehört zu den agilen Methoden. Sie ist so umfassend, dass sie häufig auch als agiles Projektmanagement-Framework bezeichnet wird. Der Fokus bei DSDM liegt auf Methoden zur Lieferung einer Softwarelösung und weniger auf zwischenmenschliche Interaktion.

[96] Vgl. Coyle und Conboy (2009), S. 145, 147.

[97] Vgl. Borgmeier (2006), S. 14; vgl. Etzel (2007), S. 37, 39.

[98] Borgmeier (2006), S. 13.

In agilen Projekten ist die Dokumentation meist zweitrangig. Der Fokus liegt auf funktionalen und schnellen Ergebnissen. Für die Qualitätssicherung und die langfristige Dokumentation von Designentscheidungen sind Spezifikationen notwendig. Ebenfalls gibt es in agilen Projekten keinen separaten Prozess zur Qualitätssicherung. Die Komponenten- und Akzeptanztests (siehe Kap. 5.2.6) werden von den Entwicklern ausgeführt. Die häufig angeführten User-Stories beschreiben lediglich das Verhalten der Software und nicht das Design. Die Dokumentation des Designs findet im Code in Form von Kommentaren statt[99]. Ein zusätzliches Team für die Qualitätssicherung hilft, die Anzahl der möglichen Fehler in einem Softwareprodukt zu reduzieren. Zeitgleich kann dadurch der Entwicklungsprozess optimiert werden.

Qualitätskennzahlen werden in Produkt-, Prozess- und Ressourcen-Metriken unterschieden. Beim Operationalisieren von Kennzahlen ergeben sich nach Peischl und Wotawa häufig fünf Hauptprobleme[100]:

1. Oft ist unklar, welche Messdaten zu erfassen sind, um bestimmte Qualitätsattribute zu bewerten. Die Daten werden unstrukturiert ohne Verknüpfung mit einem Qualitätsziel angeboten. Daher kann in einem *lean* Projekt auch kein TQM- oder GQM-Ansatz für die Kennzahlenerfassung genutzt werden. Die notwendigen Daten lassen sich nicht mit vertretbarem Aufwand operationalisieren.
2. Die gesammelten Daten werden zusammenhanglos angeboten. Die Ermittlung und Verfolgung von Abweichungen und deren Ursachen ist daher nicht möglich.
3. Die Daten bzw. die darauf errechneten Kennzahlen werden nicht in ein einheitliches Gesamtbild eingegliedert, was die Kontrolle konkreter Qualitätsattribute erschwert.
4. Werkzeuge zur Erfassung von Kennzahlen für Softwarequalität sind nur prototypisch vorhanden.
5. Das Benchmarking von Projekten ist kaum möglich, weil die dafür relevanten Kennzahlen im operativen Projektgeschäft nicht automatisch ermittelbar sind.

3.6 Zusammenfassung

Tabelle 3.1 fasst die Erkenntnisse aus den vorangegangenen Ausführungen zusammen. Vorgehensmodelle bilden den Kern der Softwareentwicklung. Die Mehrzahl der Modelle sind für Festpreisprojekte ungeeignet. Aufgrund der unterschiedlichen Rahmenbedingungen, Zielsetzungen und Inhalte von IT-Vorhaben in der Softwareentwicklung ist es nicht möglich, ein Vorgehensmodell verbindlich vorzuschreiben. Das geeignete Modell sollte bereits vor Projektstart sorgfältig ausgewählt werden.[101]

[99] Vgl. Stephens und Rosenberg (2003), S. 229–238.

[100] Vgl. Peischl und Wotawa (2010), S. 67, 69 f.

[101] Vgl. Wieczorrek und Mertens (2008), S. 65.

Tab. 3.1 Übersicht der bewerteten Modelle. (Eigene Darstellung)

Kriterium	Wasserfallmodell	V-Modell	XP	Scrum
Phasenmodell	✓	✓	–	–
Skalierbarkeit	–	–*	✓	✓
Flexibilität	–	–	✓	✓
Vorgehen	Sequentiell	Sequentiell	Iterativ	Iterativ
Methoden	–	✓	✓	✓
Werkzeuge	–	✓	–	–
Unterstützende Prozesse	–	✓	–	–
Praxisnähe	–	✓	✓	✓
Risikosteuerung	–	✓	–	–
Qualitätssicherung	✓	✓	✓	✓

✓ Erfüllt,
– Nicht erfüllt
*mit Einschränkungen. Erklärung in der Einzelkritik

Steigt bei agilen Vorgehensweisen in Großprojekten die Komplexität zur Koordination der einzelnen Teams, ist eine Anpassung der Organisationskultur erforderlich, um agile Teams effizient zu führen. Agile Vorgehensweisen bzw. agile Methoden lassen sich in einen sequentiellen Kontext einbetten, um die Organisationskultur nicht umzustellen.[102] Bei der Konzeption des hybriden Modells wird dies besonders berücksichtigt.

[102] Siehe auch Stephens und Rosenberg (2003), S. 65.

Hybride Vorgehensmodelle

4

Die Ausführungen in Kap. 3.5 ff. haben gezeigt, dass weder phasenorientierte noch agile Vorgehensweisen für IT-Vorhaben mit Festpreischarakter besonders geeignet sind. Mit der Entwicklung eines hybriden Modells für die Softwareentwicklung im Rahmen eines Werkvertrags soll die Verlässlichkeit eines Festpreises verbessert werden. Dazu werden zunächst die Charaktereigenschaften des hybriden Modells formuliert.

Neben bekannten Methoden und Techniken werden im hybriden Modell neue Methoden vorgeschlagen. Ziel bei der Entwicklung des hybriden Modells ist es, einen Methodensatz für die Bearbeitung und Organisation von mittleren Vorhaben in Form von Festpreisprojekten zu liefern.

4.1 Kerngedanken des Modells

Das hybride Modell kombiniert klassische Phasenmodelle wie das Wasserfallmodell mit agilen Methoden wie sie bspw. bei Scrum verwendet werden. Ziel des Modells ist es, durch Kombination die Stärken beider Denkweisen zu nutzen und deren Schwächen zu umgehen.

Ein wesentliches Merkmal der phasenorientierten Vorgehensmodelle ist eine vollständige Erhebung aller Anforderungen des Kunden zu Beginn des Projektes. Auf Basis der ermittelten Anforderungen kann der Aufwand geschätzt und ein Festpreis für die Umsetzung definiert werden. Da der Kunde bei agilen Vorgehensweisen seine Anforderungen während der Projektlaufzeit verändern kann, ist es kaum möglich einen Festpreis zu vereinbaren, wodurch die Terminierung und Leistungsverrechnung des Projekts gefährdet werden kann.

Im phasenorientierten Vorgehensmodell wird die Software erst nach der vollständigen Umsetzung von Kunden getestet. Wenn das umgesetzte Produkt nicht den Wünschen des

Gregor Sandhaus et al., *Hybride Softwareentwicklung,* Xpert.press,
DOI 10.1007/978-3-642-55064-5_4

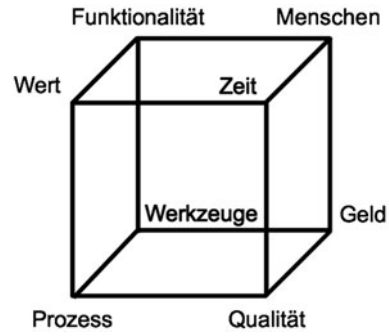

Abb. 4.1 Würfel der Beschränkungen. (Eigene Darstellung in Anlehnung an: Appelo (2011), S. 225)

Kunden entspricht, wird dies erst spät im Entwicklungsprozess festgestellt. Änderungen an der Software sind zu diesem Zeitpunkt nur mit erheblichem Mehraufwand möglich. Agile Methoden bieten in dieser Hinsicht den Vorteil des regelmäßigen Kundenfeedbacks. Sollte die Umsetzung nicht den Wünschen des Kunden entsprechen, dann fällt dies frühzeitig auf. Auf diese Weise werden teure Fehlentwicklungen vermieden.

Im Wesentlichen stellt sich der Ablauf des hybriden Modells folgendermaßen dar:

In einer sequentiellen Vorbereitungsphase werden in Zusammenarbeit mit dem Kunden die Anforderungen an die Software erhoben und vertragliche Rahmenwerte definiert. Wenn sich beide Vertragsparteien geeinigt haben, folgt die Entwicklungsphase. In dieser wird zunächst das Systemdesign festgelegt. Anschließend erfolgt die Umsetzung der Anforderungen in Iterationen. Am Ende jeder Iteration wird das Produkt vom Kunden getestet. So werden evtl. Abweichungen von den definierten Anforderungen bzw. den tatsächlichen Wünschen des Kunden zeitnah identifiziert und Fehlentwicklungen vermieden. Nach Umsetzung aller Anforderung folgt die sequentielle Finalisierungsphase. In dieser findet die Kundenabnahme statt und die Auslieferung wird vorbereitet.

Dieses Vorgehen ermöglicht durch die ausführliche Anforderungsanalyse vor der eigentlichen Umsetzung die Vereinbarung eines Festpreises für die definierten Anforderungen. Durch das iterative Vorgehen in der Entwicklungsphase und das damit verbundene, zeitnahe Kundenfeedback, werden Fehlentwicklungen vermieden.

4.2 Anforderungen an das Modell

Die Beschränkungen von Projekten wurden bereits in Abb. 2.1 auf Seite 9 in Form des *Triangle of constraints* bzw. als Quadrat dargestellt. Allerdings spiegeln diese die in den vorangegangenen Kapiteln beschriebene Komplexität von IT-Vorhaben nicht ausreichend wider. Daher werden die Beschränkungen um Größen erweitert, die durch ein hybrides Modell kontrolliert werden (vgl. Abb. 4.1).

Die Dimension Nutzen wird im Würfel in die Dimensionen Funktionalität und Wert aufgegliedert. Mit Wert ist der Geschäftswert (Business Value) gemeint, der mit den implementierten Funktionen erreicht wird. Die Größe Ressourcen gliedert sich auf in Menschen, Geld und Werkzeuge. Als zusätzliche Dimension kommt der Prozess hinzu. Dieser findet in dem ursprünglichen Dreieck der Projektdimensionen keine Beachtung. Die vorangegangenen Kapitel haben allerdings herausgestellt, dass die Art des Vorgehens (Prozess) einen direkten Einfluss auf die anderen Projektdimensionen hat[1].

Im nachfolgenden Kapitel wird unter Berücksichtigung der genannten Dimensionen ein neues hybrides Modell vorgestellt. Dafür werden agile und phasenorientierte Modelle kombiniert und Methoden und Werkzeuge zur Steuerung vorgestellt. Untermauert werden die Thesen der Autoren durch Praxisbeispiele.

4.3 Mensch und Rollen

Agile Vorgehen stellen Menschen und ihre Interaktion in den Vordergrund. Steht der Mensch bei der Softwareentwicklung im Vordergrund, so kann dem Verlust von Schlüsselpersonen entgegen gewirkt werden. Die Bedürfnisse und Fähigkeiten des Teams dominieren dann das Vorhaben und nicht harte Kennzahlen aus dem Controlling. Durch die entsprechende Wertschätzung für die eigene Arbeit, bleiben Leistungsträger einem Projekt erhalten.

Allerdings kennen agile Teams keine formalen Rollen, wie sie bspw. in der Organisationslehre von Projektmanagementmethoden nach PMBOK, PRINCE2 oder ICB beschrieben werden (dazu gehören unter anderem Risikomanager, Qualitätsmanager und Controller). Die agilen Teams arbeiten interdisziplinär. Die organisatorische Verankerung der Qualitätssicherung aus den sequentiellen Modellen soll im hybriden Modell nicht verloren gehen. Darum bleiben die verantwortlichen Rollen für die Qualitätssicherung bestehen (vgl. Kap. 3, Abschn. 3.5.8.2)[2].

Grundsätzlich werden für das Projekt- und Risikomanagement Personen ausgewählt, die entsprechende Fach-, Methoden- und Sozialkompetenz besitzen. Dies ist besonders wichtig, wenn Entscheidungen unter Unsicherheit zu treffen sind, weil dann Erfahrungen und Methodenkompetenz notwendig sind, die sich Mitarbeiter nicht innerhalb kurzer Zeit aneignen können.

Darum werden nach dem hybriden Modell bei mittleren bis großen IT-Vorhaben die folgenden Rollen mit erfahrenen Personen besetzt:

- Projektleiter
- Programmierer/Entwickler
- Qualitätssicherungsbeauftragter
- Controller

[1] Siehe auch Appelo (2011), S. 225.

[2] Vgl. Ekssir (2013), S. 11.

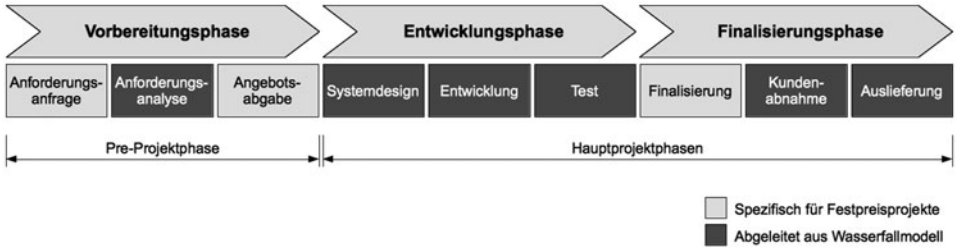

Abb. 4.2 Grundgerüst des hybriden Modells. (Eigene Darstellung in Anlehnung an: Blum und Wörsdörfer (2013), S. 54)

Mehrere Rollen können von einer Person besetzt sein oder eine Rolle durch mehrere Personen wahrgenommen wird. Bei Entscheidungsfunktionen wie dem Projektleiter, dem Qualitätssicherung-Beauftragen oder dem Controller sollte klar kommuniziert sein, welche Personen bei Entscheidungen verantwortlich sind.

4.4 Grundlegendes Phasenmodell

Im Folgenden wird das Phasen- oder Prozessmodell beschrieben, auf dem das hybride Modell basiert. Dabei werden Ideen verschiedener Unternehmen aufgegriffen, die dieses Prozessmodell in der Praxis erprobt und auf verschiedene Projektgrößen erfolgreich angewendet haben[3].

Große Vorhaben bzw. Projekte wurden früher in der Regel mit einem sequenziellen Vorgehen (bspw. Wasserfallmodell, V-Modell oder RUP) realisiert. Je nach Komplexität und Aufwand sehen auch Anhänger agiler Entwicklungsmethoden die Notwendigkeit eines klaren, regelnden Rahmens und vorgelagerten Prozessschritten zur Organisation des Vorhabens[4]. Aus der Notwendigkeit bei mittleren Vorhaben in Form von Festpreisprojekten, zusätzlich steuernde Phasen im Vorfeld zu integrieren, ergibt sich das in Abb. 4.2 dargestellte rudimentäre Prozessmodell.

Das Modell ist an das Prozessmodell der Firma Tideum (TADP-Modell) angelehnt. Das Tideum Agile Development Process Modell wird sowohl für große als auch kleine Projekte gleichermaßen eingesetzt. Es besteht aus den drei Hauptphasen Vorbereitung, Entwicklung und Finalisierung. Beim TADP-Modell sollten bestimmte Phasen abgeschlossen sein, bevor die nächste Phase beginnt. So sind im Rahmen der Anforderungsanalyse alle Vorstellungen des Kunden vollständig abgeklärt, bevor mit dem Systemdesign begonnen wird. Um Projekte mit den klassischen Methoden und dem Kontrollkriterium Planungstreue planen und steuern zu können, ist eine Abgrenzung in Hauptphasen notwendig. Kosten und Umfang eines Vorhabens lassen sich so besser abschätzen. Jede der drei Hauptphasen ist

[3] Vgl. Blum und Wörsdörfer (2013), S. 53.

[4] Vgl. Schwaber (2004); siehe auch Hunt und Thomas (2003).

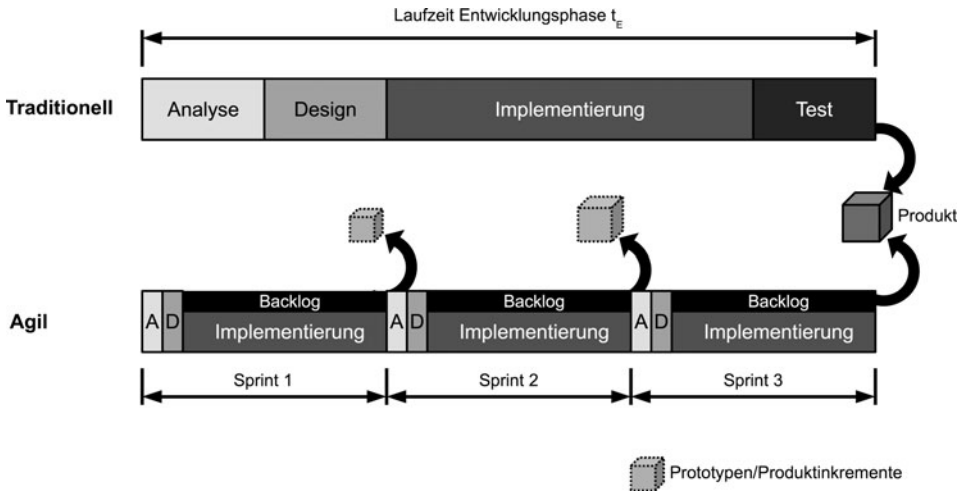

Abb. 4.3 Agiles vs. phasenorientiertes Modell. (Eigene Darstellung in Anlehnung an: Ditze et al. (2013), S. 24)

wiederum in drei Unterphasen unterteilt, die entweder direkt den Entwicklungsprozess oder administrative Schritte (Angebotsabgabe, Kundenabnahme) betreffen.[5]

Das Modell ist dem erweiterten Wasserfallmodell sehr ähnlich (vgl. Abb. 4.3). Sollen aber Ergebnisse zu einem bestimmten Zeitpunkt für den Kunden fassbar sein, dann ist in der Entwicklungsphase eine feste Taktung für das Vorhaben erforderlich. Die Detailplanung der Aktivitäten unterliegt dann einem festen Zeitraster, wie dies in den agilen Methoden XP oder Scrum in Form der Sprints postuliert wird. Am Ende jedes Taktes liegt ein prüfbares Ergebnis vor[6].

Durch dieses Vorgehen gewinnt das Projekt an Effektivität. Am Anfang wird nur ein Kern mit der geringsten notwendigen definierten Funktionalität bereitgestellt und angewendet. Durch die frühe Bereitstellung und das entsprechende Feedback der Anwender können bei großen Projekten die Termine und Kosten überschaubar gehalten und die Gesamtkomplexität des Vorhabens reduziert werden. Zusatzfunktionen und neue Funktionen werden erst akzeptiert und implementiert, wenn die Stakeholder ausreichend Erfahrung mit dem Kernsystem gesammelt haben.[7]

Um den Erfahrungsaustausch zwischen Anwendern und Entwicklern zu unterstützen, wird das TADP-Modell innerhalb der Entwicklungsphase nach einem an Scrum angelehnten Vorgehen durchgeführt[8]. Dadurch wird der Kunde nach der Anforderungsanalyse weiter in den Prozess der Erstellung involviert und das Entwicklungsteam kann flexibler

[5] Vgl Blum und Wörsdörfer (2013), S. 53 f.

[6] Vgl. Henkel et al. (2011), S. 65.

[7] Vgl. ebd., S. 67 f.

[8] Vgl. Blum und Wörsdörfer (2013), S. 54 f.

auf Änderungen reagieren. An dieser Stelle besteht aber auch die Gefahr, dass Scrum falsch angewendet wird. Laut dem Miterfinder von Scrum, Ken Schwaber, darf sich ein Prozess nur dann Scrum nennen, wenn auch alle Regeln des Modells streng befolgt werden[9]. Bereits vorher wurde jedoch dargestellt, dass das strenge Befolgen von definierten Regeln eines Modells durch die Unternehmenskultur nicht möglich ist. Aus diesem Grund ist der Entwicklungsprozess generell nach einem iterativen Vorgehen organisiert und innerhalb des Prozesses werden anerkannte agile Methoden genutzt werden, die die Flexibilität des Teams unterstützen.

In den Kap. 3.4 ff. wurde gezeigt, dass das iterative Vorgehen in den meisten agilen Modellen identisch ist; die Methoden gleichen sich im Kern und differieren lediglich im Detail. Daher lässt sich für agile Vorgehensweisen ein Phasenmodell wie in Abb. 4.2 auf Seite 57 ableiten.

Die gesamte Projektlaufzeit wird in Sprints unterteilt. Am Ende eines jeden Sprints gibt es einen Prototypen oder ein Produktinkrement, das dem Kunden zum Test oder zur Nutzung übergeben wird. Üblicherweise haben Sprints im hybriden Modell eine Dauer von zwei bis vier Wochen[10]. Im Rahmen von Planungstreffen mit Kunden erfolgt zu Anfang eines Sprints die Analyse und das Design für die im aktuell anstehenden Sprint umzusetzenden Anforderungen[11]. Geänderte Anforderungen während eines Sprints werden im Backlog festgehalten. Zur Aktivität der Implementierung gehört nach agilen Maßstäben auch der Test.

4.5 Umgang mit äußeren Einflussfaktoren

Bei der Erstellung eines Vorgehensmodells sind äußere Einflussfaktoren zu berücksichtigen. In diesem Kapitel wird daher das Zusammenspiel des hybriden Modells mit den wesentlichen äußeren Einflussfaktoren betrachtet.

4.5.1 Berücksichtigung von IT-Compliance

IT-Compliance als Einflussfaktor (siehe Kap. 2.10) setzt voraus, dass Unternehmen im Geschäftsbetrieb nicht gegen Gesetze und selbstauferlegte Regelungen sowie die guten Sitten verstoßen.

Die Einhaltung von IT-Compliance Vorschriften kann nicht allgemein betrachtet werden, da sie sich für verschiedene Unternehmen aus verschiedenen Branchen unterscheiden. In den Fallstudien (siehe Kap. 6 und 7) wird darum speziell auf die Berücksichtigung von

[9] Vgl. Schwaber (2004).

[10] In Kap. 5.1 ist die Berechnung einer Sprintdauer ausführlich dargestellt.

[11] Vgl. Ditze et al. (2013), S. 24.

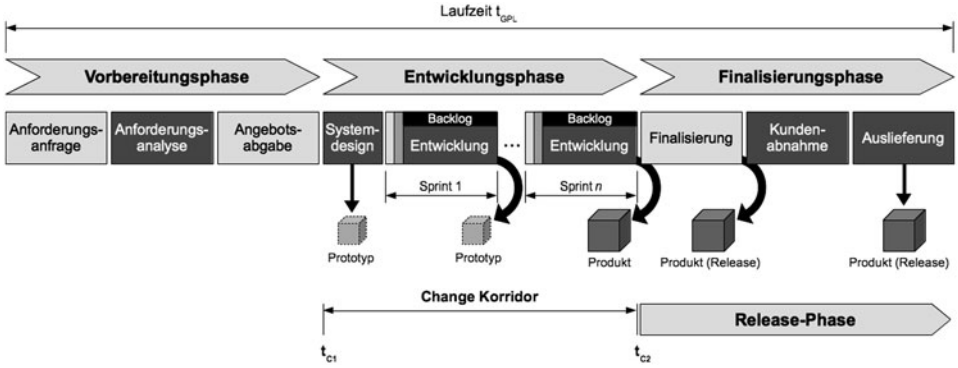

Abb. 4.4 Zusammenspiel des hybriden Modell mit ITIL™. (Eigene Darstellung)

IT-Compliance eingegangen. Grundsätzlich empfiehlt es sich, die Anforderungen an die IT-Compliance vorab zu identifizieren und zu bewerten.

4.5.2 Einbindung von ITIL™ -Prozessen

Ist das hybride Modell in den Kontext eines professionellen IT-Betriebs, der nach dem ITIL™ -Framework organisiert ist, eingebettet, so hat das Modell die Prozesse im Rahmen der Service Transition zu berücksichtigen.

Wie in Kap. 2, Abschn. 2.11.2 beschrieben, haben vor allem die Prozesse der Service Transition Phase des ITIL™ Service Life-Cycles Auswirkungen auf die Softwareentwicklung. Als wesentlich wurden die Prozesse „Change Management“ und „Release and Deployment Management“ identifiziert. Beide Prozesse lassen sich ohne großen Aufwand in das hybride Vorgehensmodell integrieren.

Gemäß ITIL™ soll jede Änderung an einem IT-System den Change-Prozess durchlaufen. Um den Change-Prozess anzustoßen, wird ein Request for Change (RfC) erstellt, der dann die Grundlage für die Bewertung und Priorisierung durch den Change Manager ist.

ITIL™ selbst trifft keine Aussage darüber, zu welchem Zeitpunkt der Change Request zu erstellen ist. Je früher dies geschieht umso mehr Zeit steht dem Change Manager vor dem gewünschten Release-Termin für die geplanten Prüfungen zur Verfügung. In der Regel geschieht dies während der Entwicklungsphase. In Abb. 4.4 ist dieser Zeitraum durch die Punkte t_{C1} und t_{C2} dargestellt

Ein Release wird hingegen erst in der Finalisierungsphase gebildet. Für jede geplante Auslieferung wird ein Release erstellt.

Wenn die Software nicht ausgeliefert sondern zum Test bereitgestellt wird, dann ist es nach ITIL™ nicht notwendig, ein Release zu erstellen. Somit entfallen für die Bereitstellung der Software zum Test in den einzelnen Iterationen der Entwicklungsphase die Erstellung der RfCs und der Releases.

4.6 Messung und Kennzahlen

Um die Zielerreichung zu überprüfen, werden Kennzahlen definiert und diese sowohl vor und nach der Einführung erhoben und miteinander verglichen.

Im Rahmen der beiden Fallstudien (vgl. Kap. 6 und 7) werden Kennzahlen zur Bestimmung der Flexibilität und der Durchlaufzeit definiert und Verfahren zu deren Erhebung erläutert. Anschließend werden weitere Kennzahlen vorgestellt, die zur Bewertung der Teamleistung herangezogen und im vertraglichen Umfeld in Projekten genutzt werden.

4.6.1 Erfolge richtig messen

Basten et al. zeigen anhand empirischer Forschungsergebnisse, dass das *Iron Triangle* (Abb. 2.1) keine adäquaten Faktoren bietet, um den Gesamterfolg von IT-Vorhaben in der Softwareentwicklung zu messen. Die alleinige Konzentration auf Planungstreue (Termintreue, Budgettreue, Anforderungserfüllung und Qualität) ist kein hinreichendes Kriterium, um den Erfolg von Projekten zu messen. Z. B. sind Kunden- bzw. Stakeholderzufriedenheit weitere Erfolgsfaktoren. Zusätzlich kann die Bewertung des Erfolgs mehrere Phasen berücksichtigen und nicht nur die Implementierungsphase. Den Erfolg über den gesamten Produktlebenszyklus zu ermitteln, ist allerdings sehr weit gegriffen und nicht auf alle Arten von IT-Vorhaben anwendbar. Grundsätzlich sollten bei der Erfolgsbetrachtung das Produkt und der Entwicklungsprozess einbezogen werden.[12]

Erfolg lässt sich mit den betriebswirtschaftlichen Konzepten Effektivität und Effizienz messen. Effizienz bezieht sich im Modell von Basten et al. auf den Prozess und die Effektivität auf das erstellte Produkt. Ein Entwicklungsprozess ist demnach erfolgreich, wenn er dem ökonomischen Prinzip genügt. Das bedeutet, dass er in Bezug auf Ressourcenverbrauch und Leistungserstellung optimal ist. Da Erfolg von den Stakeholdern eines Projekts unterschiedlich wahrgenommen wird, werden Effizienz und Effektivität in dem Modell separat betrachtet.[13]

Prozesseffizienz und Mitarbeiterzufriedenheit sind laut Basten et al. mit dem Entwicklungsprozess assoziierte Größen, während sich die Kundenzufriedenheit eher auf das erstellte Produkt beschränkt.[14]

IT-Vorhaben werden laut der Studie nicht notwendigerweise als erfolgreich wahrgenommen, wenn nur Planungstreue vorliegt. Planungstreue hat einen geringen Einfluss auf den wahrgenommenen Gesamterfolg. Dies steht im Gegensatz zu den steuernden und lenkenden Kenngrößen für IT-Vorhaben in Form von Projekten. Planeinhaltung ist das wesentliche Instrument zur Fortschrittskontrolle. Vergleichbares gilt für die Anforderungserfüllung. Wichtiger scheint den Studienteilnehmern, „wenn zufriedene Mitarbeiter einen effizienten

[12] Vgl. Basten et al. (2011), S. 7 f.

[13] Vgl. ebd., S. 8.

[14] Ebd., S. 14–16.

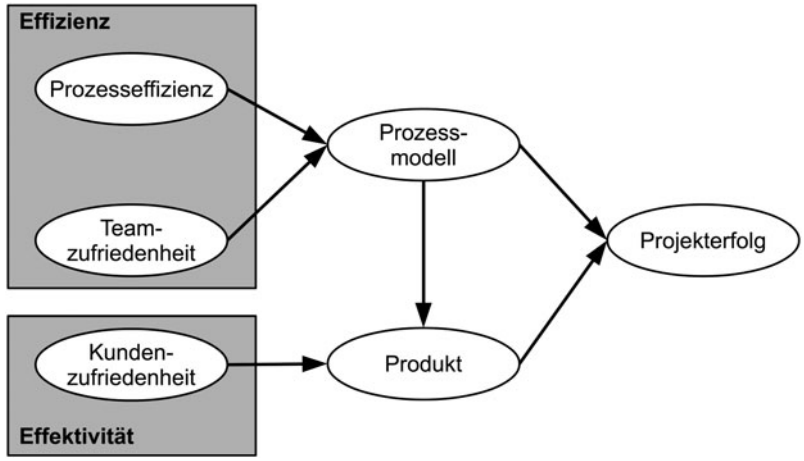

Abb. 4.5 Angepasstes Modell des Erfolgs. (In Anlehnung an: Basten et al. (2011), S. 14–16)

Prozess durchführen und der Kunde am Ende mit dem Ergebnis des Projekts zufrieden ist.“[15] (vgl. Abb. 4.5).

4.6.2 Kennzahlen für die Erfolgsmessung

Messtechnische Bewertungen des Projekterfolgs sind auch in vertraglicher Hinsicht sinnvoll. Zum Einen möchten die Stakeholder Auskunft über den Projektfortschritt erhalten, zum Anderen wünscht der Auftragnehmer auch die Leistung seines Teams bzw. einzelner Mitglieder zu bewerten. Wie hoch der Aufwand zur Erhebung der notwendigen Kennzahlen ist, hängt auch von der Werkzeugunterstützung ab. Eine pauschale Aussage kann darum nicht getroffen werden. Je besser die Werkzeugunterstützung und je erfahrener der Projektleiter, desto weniger Aufwand ist für die Erhebung der Kennzahlen erforderlich.

Aus der agilen Methodik lassen sich vier Kennzahlen ableiten. Diese basieren auf der Verbreitung in agilen Projekten und lassen sich auf iterative Vorgehen übertragen[16]:

- **Technical Debt Points:** Bewertung der qualitativen Projektentwicklung anhand der Menge offener architektonischer und quellcodebezogener Problembereiche. Typische Problembereiche beziehen sich auf fehlende Dokumentation, hohe Abhängigkeiten zwischen den Systemkomponenten oder nicht eingehaltene Namenskonventionen. Die Problembeseitigung erfolgt z. B. durch das Refactoring. Angestrebt wird eine geringe Anzahl offener Probleme.

[15] Basten et al. (2011), S. 16.
[16] Vgl. ebd., S. 65.

- **Enhanced Burndown Chart:** Bei dieser Kennzahl handelt es sich um die Bewertung der je Iteration umgesetzten Anforderungen unter Berücksichtigung der Realisierungsgeschwindigkeit (velocity) und ggf. veränderten Zielstellungen (scope change). Mit Hilfe dieser Kennzahlen kann die Fertigstellung weiterer Anforderungen abgeschätzt werden.
- **Business Value Delivered:** Jeder ausgelieferten Anforderung aus dem Backlog wird ein Geschäftswert zugeordnet. Typischerweise werden hier möglichst hohe Werte angestrebt.
- **Story Points:** Die durch den Auftraggeber bereits akzeptierten User Stories werden mit Story Points bewertet. Ein Story Point ist eine relative Maßeinheit zur Bewertung der Komplexität oder Größe einer Anforderung. Nach einigen Sprints sind agile Teams in der Lage abzuschätzen, wie viele Story Points sie im nächsten Sprint umsetzen können.

Mit Hilfe dieser Messansätze lässt sich der Projekterfolg am Ende ermitteln. Es steht weniger die Planungstreue im Vordergrund als die Unterstützung des Auftragnehmers zur Erreichung der Geschäftsziele. Diese zusätzlichen Messansätze werden zu den typischen klassischen Kennzahlen erhoben. Dazu gehören[17]:

- Änderungsdichte
- Fortschrittsgrad
- Ressourcenauslastung
- Rentabilität des Projekts
- Wirtschaftlichkeit des Projekts

Die Änderungsdichte misst die Anzahl der Änderungsanforderungen über einen bestimmten Zeitraum. Wird zusätzlich zwischen angenommenen und abgelehnten Anforderungen unterschieden, ist diese Kennzahl ein Indikator für die Qualität der Anforderungen. Eine steigende Anzahl neuer, akzeptierter Anforderungen ist ein Indikator dafür, dass Zeitplan und Projektbudget gefährdet sind.

[17] Vgl. Kütz (2009).

Hybrides Vorgehen für Festpreisprojekte

5

Das Grundgerüst eines hybriden Modells kann für viele Projektarten verwendet werden. Das folgende Kapitel zeigt auf, wie ein modifiziertes Modell auch für Festpreisprojekte genutzt werden kann.

5.1 Hybrides Phasenmodell

Für Projekte wird das Grundgerüst des hybriden Modells aus Kap. 4.5 um unterstützende Prozesse erweitert, die das Modell auch für sicherheitskritische Anwendungen (z. B. Automobiltechnik, medizinische Systeme oder Anwendungen in der Luftfahrt) anwendbar machen. Das modifizierte Modell ist in Abb. 5.1 dargestellt.

Die **Vorbereitungsphase** dient im hybriden Modell der Erstellung eines verlässlichen Angebots auf Basis des Lastenheftes des Auftraggebers. Dabei ist zu berücksichtigen, dass während der **Anforderungsanalyse** Rückfragen an den Auftraggeber gestellt werden, um fehlende Anwendungsfälle oder nicht vollständig dargestellte Fachobjekte genauer zu spezifizieren. Bei öffentlichen Ausschreibungen findet die Anforderungsanalyse häufig in einer mehrwöchigen Fragerunde statt. Potenzielle Bewerber haben in dieser Zeit die Möglichkeit, die Ausschreibung zu analysieren und Fragen zur Klärung offener Punkte zu stellen. Diese Fragen und die entsprechenden Antworten werden vom Auftraggeber an alle Bewerber weitergegeben. Auf Basis der Anforderungsanalyse wird die Kosten- und Aufwandsschätzung erstellt. Sofern nicht eine Projektdatenbank mit historischen Daten vergleichbarer Projekte vorliegt, werden die Schätzungen mit Hilfe des Use-Case- oder Object-Point-Verfahren vorgenommen. Ältere Methoden wie Constructive Cost Model (COCOMO) oder Function Point sind bei Individualentwicklungen oder innovativen Projekten nicht hilfreich, da hier die notwendigen historischen Daten für eine adäquate

Gregor Sandhaus et al., *Hybride Softwareentwicklung,* Xpert.press,
DOI 10.1007/978-3-642-55064-5_5

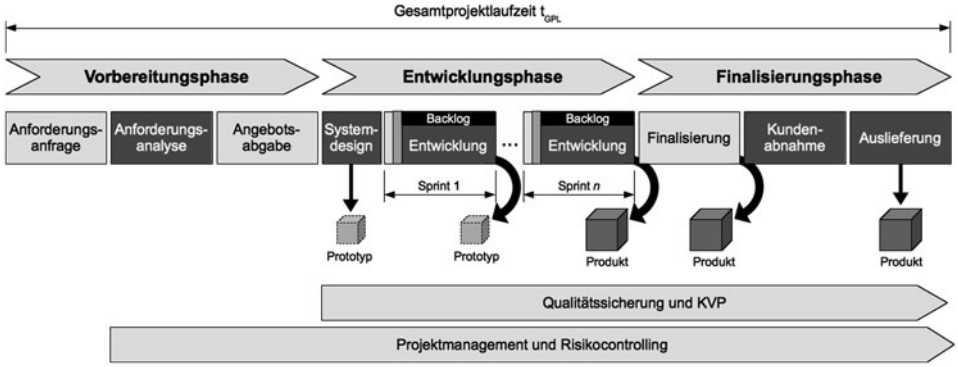

Abb. 5.1 Hybrides Modell für projektorientierte Vorhaben. (Eigene Darstellung)

Aufwandsschätzung nicht oder nur teilweise vorliegen. Bei der Kosten- und Aufwandsschätzung sind auch die Zeiten für die Anforderungsanalyse und die Angebotserstellung zu berücksichtigen. Diese Kosten werden bei Festpreisprojekten selten separat geltend gemacht und werden darum in den Aufwand für die Entwicklung einkalkuliert. Somit setzt sich die Aufwandsschätzung der Gesamtprojektlaufzeit t_{GPL} aus der Zeit für die Vorbereitungsphase t_V, die Entwicklungsphase t_E und die Finalisierungsphase t_F zusammen:

$$t_{GPL} = t_V + t_E + t_F$$

Da sich der Aufwand für die Vorbereitungsphase im Vorfeld nicht genau mit Hilfe der Aufwandsschätzmethoden fakturieren lässt, wird die Angabe für t_V im Verhältnis zur Entwicklungszeit t_E angegeben. Zusätzlich kann die Komplexität k des Projekts in Form eines Komplexitätsexponenten für die Vorbereitungsphase berücksichtigt werden. Der Komplexitätsexponent orientiert sich an der Aufwandsschätzung nach dem COCOMO-Verfahren. Somit ergibt sich mit $t_V = t_E^k$:

$$t_{GPL} = t_E^k + t_E + t_F$$

Eine Herausforderung ist die Bestimmung des Komplexitätsexponenten k. Die in COCOMO verwendeten, statistisch gewonnen Faktoren für die Bestimmung der Komplexität von Projekten können für die Vorbereitungsphase von Festpreisprojekten nicht angewendet werden. Es ist also notwendig, diese Faktoren empirisch aus historischen Daten zu ermitteln. Dies sollte jedes Unternehmen individuell vornehmen. Kap. 5, Abschn. 5.2.2 betrachtet diese Aufwandsschätzung detaillierter.

Kommt der Vertrag zustande, so beginnt die Entwicklungsphase. Die **Entwicklungsphase** ist iterativ und inkrementell aufgebaut. Am Anfang steht das grobe **Systemdesign**, an dessen Ende einer oder mehrere Prototypen stehen. In dieser Phase wird auch über den Einsatz zu verwendender Technologien (z. B. die Programmiersprache) entschieden. In

Tab. 5.1 Empfohlene Dauer einer Iteration anhand der Vorhabengröße. (Quelle: eigene darstellung)

Klassifikation	t_i in Werktagen
Klein	10
Mittel	10
Groß	15
Sehr groß	20

einer festgelegten Anzahl Sprints wird das Produkt über mehrere Prototypenstadien entwickelt. Die Prototypen dienen zur Herleitung eines gemeinsamen Konsenses zwischen allen Projektbeteiligten. Durch die iterative Entwicklung kann der Auftraggeber am Ende jeder Iteration (Sprint) einen Prototypen bzgl. seiner Anforderungen testen und prüfen. Dazu empfiehlt sich auf Seiten des Auftraggebers die Einplanung entsprechender Ressourcen, die im Vorfeld das zukünftige Produkt testen.[1]

Für die Dauer einzelner Iterationen im hybriden Modell empfiehlt sich in der Regel wie bei Scrum eine Dauer zwischen zwei und vier Wochen (vgl. Tab. 5.1). Ob nach einer Iteration die Auslieferung an den Kunden erfolgt, hängt von der Gesamtprojektlaufzeit und der Komplexität des Projekts, dem Fortschrittsgrad und den benötigten Aufwänden für eine Auslieferung ab. In dieser Zeit können verschiedene Methoden angewendet werden, die die Entwickler bei der Arbeit und der Qualitätssicherung dahingehend unterstützen, dass zu jeder Zeit ein auslieferbarer Prototyp für den Kunden gebaut werden kann (siehe auch Kap. 5, Abschn. 5.2.4). Für die Bestimmung der Iterationen gilt, dass sich gesamte die Entwicklungsphase t_E zusammensetzt aus der Zeit für eine Iteration t_i und der geplanten Anzahl der Iterationen n:

$$t_E = t_i \cdot n$$

Dabei sollte für die Dauer einer Iteration berücksichtigt werden, dass eine Iteration umso länger dauert, je komplexer das Projekt ist. Ausgehend von einer Arbeitswoche mit fünf Werktagen und der Klassifikation der Projektgröße aus Kap. 2.3 können die Festlegungen für die Dauer einer Iteration, wie in Tab. 5.1 dargestellt, getroffen werden.

Somit lässt sich die Anzahl der Iterationen als Quotient der gesamten Entwicklungsdauer und der empfohlenen Dauer einer Iteration anhand der Tabelle bestimmen:

$$n = t_E / t_i$$

Die Dauer der Iterationen sollte über die komplette Projektlaufzeit konstant gehalten werden, um eine gleichmäßige Taktung über die Gesamtlaufzeit zu erhalten. So weiß jedes Projektmitglied, wann die Arbeit abgeliefert werden soll.

In der **Finalisierungsphase** erfolgen die Integrationstests. Hier werden die notwendigen Dokumentationen erstellt oder vervollständigt und das Produkt letzten Tests unter realen

[1] Vgl. Sneed (2004), S. 36.

Einsatzbedingungen (Fabriktest) unterzogen. Anschließend erfolgt die Kundenabnahme. In dieser Zeit kann der Kunde noch Nachbesserungen seiner Anforderungen und die Behebung von Defects, die die Inbetriebnahme gefährden, verlangen. Defects, die im Anschluss an die Auslieferung gefunden werden, sind vom Lieferanten innerhalb der Gewährleistungsfrist zu beheben. Da Software niemals fehlerfrei ist, sollte dies bei der Angebotskalkulation ebenfalls berücksichtigt werden, da sich die Defects auf die Rendite des Projekts auswirken.

Zusätzlich zu den drei Hauptphasen besitzt das Vorgehensmodell vier unterstützende Prozesse, die zu zwei Prozessen aggregiert werden. Dies hängt mit der Rollenvergabe im Projekt zusammen. Die Personen, die für die Qualitätssicherung zuständig sind, sollten auch den kontinuierlichen Verbesserungsprozess (KVP) betreuen, da sich beide Prozesse ergänzen. Projektmanagement und Risikocontrolling sind bei Festpreisprojekten so eng verbunden, dass diese ebenfalls durch die Rolle des Projektleiters übernommen werden sollten. Auf die unterstützenden Prozesse wird in Kap. 5.4 detaillierter eingegangen.

5.2 Methoden

Im Folgenden werden die wichtigsten Methoden für die einzelnen Phasen innerhalb der Gesamtprojektlaufzeit vorgestellt. Hierbei handelt es sich um Methoden aus sequentiellen und agilen Vorgehen, die in das hybride Modell integriert sind. Jeder der drei Hauptphasen werden verschiedene Methoden zugewiesen, die sich in der Praxis bewährt haben.

Die Methodenzuweisung für die einzelnen Phasengestaltet sich wie folg:

1. **Vorbereitungsphase:** Requirements Engineering (Kap. 5, Abschn. 5.2.1), Kosten- und Aufwandsschätzung (Kap. 5, Abschn. 5.2.2), Nutzenanalyse (Kap. 5, Abschn. 5.2.3)
2. **Entwicklungsphase:** Requirements Engineering (Kap. 5, Abschn. 5.2.1), Prototyping (Kap. 5, Abschn. 5.2.4), Test-Driven-Development (Kap. 5, Abschn. 5.2.5), Continuous Integration (Kap. 5, Abschn. 5.2.7)
3. **Finalisierungsphase:** Die Finalisierungsphase wird weniger stark berücksichtigt. Hier gelten die Regeln zur Auslieferung, wie sie in sequenziellen Vorgehen definiert oder vom Auftraggeber vorgegeben sind.

Über alle Phasen hinweg stehen das Qualitätsmanagement, das Projektmanagement und das Risikocontrolling. Auf die in diesen unterstützenden Prozessen zu verwendenden Methoden ist bereits in Kap. 3, Abschn. 3.5.8 eingegangen worden. In Kap. 5.4 werden diese für das hybride Modell aufgegriffen und vertieft.

5.2.1 Requirements Engineering

Wie wichtig die Anforderungsanalyse ist, zeigt die Aussage von Avci: „Missverständliche, inkonsistente und fehlende Anforderungen machen 67 % der Anforderungsfehler aus. . .“[2] Zu den Ursachen gehören die Art der Dokumentation, die Form der Beteiligung der Stakeholder, das Wissen der Entwickler über das Anwendungs- bzw. Sachgebiet und die Art der Erhebung der Anforderungen.

Im klassischen Requirements Engineering (RE) spielt der Analytiker eine wesentliche Rolle. Seine Aufgabe ist es, aus den oft diffusen und widersprüchlichen Wünschen der Stakeholder eine systematische Anforderungsspezifikation zu erarbeiten. Diese Spezifikation besteht aus Use-Cases, Objektmodellen, Prototypen und den modellierten Geschäftsprozessregeln. Agiles RE kennt dies alles nicht. In einem Großteil der agilen Projekte dienen User-Storys als alleinige Anforderungsspezifikation[3]. Göbl und Köhler zeigen basierend auf den Arbeiten anderer Autoren, dass sich agiles und klassisches RE durchaus zusammenführen lassen.

Dem Konzept liegt zugrunde, dass ein Softwareprodukt einen Geschäftsprozess abbildet. Aus dem Prozess heraus ergeben sich Wünsche der Anwender an das Softwaresystem, die in fachliche Anforderungen münden. In der klassischen RE-Praxis werden aus diesen fachlichen Anforderungen Use-Cases formuliert. Diese wiederum können durch Szenarien weiter verfeinert werden. Aus den Detailanforderungen lassen sich schließlich die Aufgaben für die Entwicklung ableiten. Vergleichbar ist das Vorgehen im agilen RE: Eine fachliche Anforderung wird als Thema erfasst und durch User-Storys weiter verfeinert. Aus der User-Story werden Tasks für die Softwareentwicklung abgeleitet. Abgesehen von den Methoden und den Begrifflichkeiten unterscheidet sich das Vorgehen nicht.[4]

Beide Arten des Vorgehens besitzen ein Konzept zur Systemabgrenzung und zum Clustering von Funktionalität. Use-Cases können iterativ entwickelt werden User-Storys lassen sich als Use-Case abbilden. Somit kann mit der Implementierung bereits begonnen werden, wenn alle Szenarien eines Use-Cases noch nicht vollständig beschrieben sind.[5] Alle weiteren Konzepte des klassischen RE wie UI-Mockups, Geschäftsregeln und Datenmodelle werden auch im agilen RE benötigt, sind dort ggf. nur anders dokumentiert oder zu einem anderen Zeitpunkt erhoben.

Die Form der Dokumentation der Anforderungen sorgt bei Entwicklern für einen bekannten Zwiespalt: Sollen Anforderungen natürlich-sprachlich oder semi-formal mit Modellen wie UML beschrieben werden?

Studien zeigen, dass der Einsatz von Modellen zu weniger Fehlern führt; andererseits haben Stakeholder jedoch häufig das Problem, die semi-formale Darstellung ihrer Anforderungen nicht zu verstehen. Darum tendieren viele Entwickler dazu, die Anforderungen

[2] Avci (2008), S. 98.

[3] Vgl. Göbl und Köhler (2013), S. 79.

[4] Vgl. ebd., S. 80.

[5] Vgl. ebd., S. 80 f.

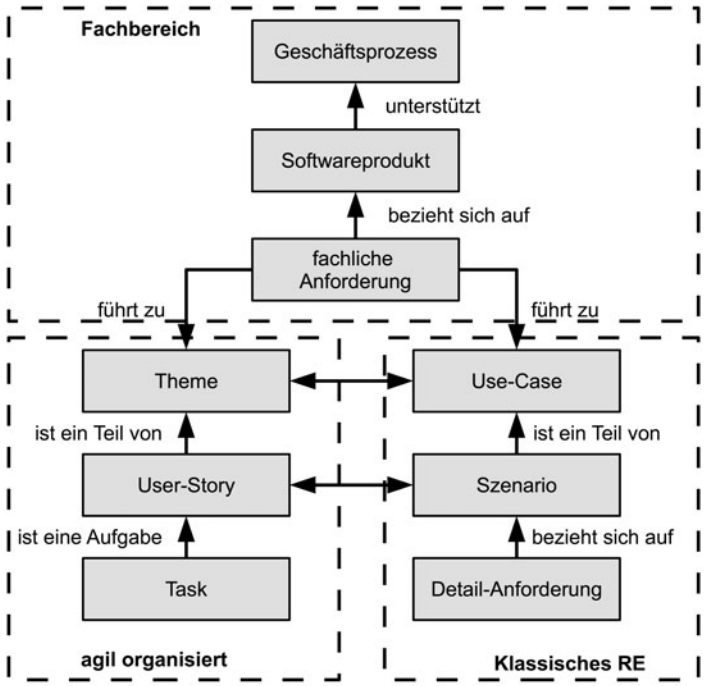

Abb. 5.2 Begriffe des agilen und klassischen RE und deren Zusammenspiel. (Eigene Darstellung in Anlehnung an: Göbl und Köhler (2013), S. 80)

semi-formal zu beschreiben und zusätzlich für die Stakeholder um natürlich-sprachliche Erläuterungen zu ergänzen.[6]

Wichtiger als die Form der Dokumentation ist allerdings das fachspezifische Wissen über das Anwendungsgebiet und die Form der Benutzerbeteiligung. Häufig spielen Benutzer und Entwickler in der Formulierung der Anforderungen nur eine Nebenrolle. Es wird nur ein Vertreter der Benutzer bestimmt, um die unzulängliche Einbindung der späteren Anwender zu kaschieren. Das bedeutet, dass die späteren Anwender eines Systems nur selten für diese Aufgabe zur Verfügung gestellt werden. Auch sollten die Entwickler Erfahrung mit dem Anwendungsgebiet haben. Anforderungsfehler werden begünstigt, wenn bei der Erfassung der Anforderungen das Wissen über die allgemeinen Geschäftsprozesse gering oder wenig ausgeprägt ist. Insbesondere kommt es somit zu der Definition von unklaren Zielen und einer unzureichenden Auseinandersetzung mit den Kundenbedürfnissen[7] (Abb. 5.2).

Unterschiedliche Stakeholder eines Software-Systems stellen unterschiedliche Anforderungen. Daraus resultiert, dass das System bei der Aufnahme und Analyse der Anforderungen aus den verschiedenen Perspektiven der Stakeholder betrachtet wird. Aus

[6] Avci (2008), S. 99.

[7] Ebd., S. 100 f.

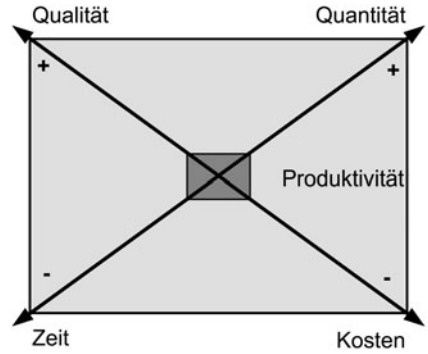

Abb. 5.3 Quadrat des Teufels. (Eigene Darstellung in Anlehnung an: Sneed (2007a), S. 55)

den Zielen der Stakeholder werden die notwendigen Aufgaben abgeleitet und als Geschäftsprozess erfasst. Die erfassten Geschäftsprozesse werden mit Fortschreiten des Entwicklungsprozesses weiter konkretisiert und in Teilprozesse, Systemfunktionen und Interaktionen zerlegt[8].

5.2.2 Kosten- und Aufwandsabschätzung

In diesem Kapitel wird eine Auswahl anwendbarer Verfahren zur Kosten- und Aufwandsabschätzung für Festpreisprojekte vorgestellt. Für Festpreisprojekte ist eine ausführliche Anforderungsspezifikation erforderlich. Im sogenannten Lastenheft sind alle wesentlichen funktionalen und nicht-funktionalen Systemmerkmale enthalten und um Anwendungsfälle, Geschäftsobjekte, Prozesse, Benutzeroberfläche und Systemschnittstellen zu ergänzen. Andernfalls ist eine Zählung bzw. Schätzung nach den bekannten Aufwandsschätzmethoden nicht möglich.[9]

In Abb. 5.3 sind die fünf Kernmetriken – Quantität, Qualität, Zeit, Kosten und Produktivität – für die Softwareentwicklung dargestellt. Sie bilden die Basis für alle Aufwandsschätzmethoden. Die Produktivität ist dabei eine Konstante. Sie ist zu einem bestimmten Zeitpunkt für eine Organisation gegeben und geht aus den Erfahrungen mit bisherigen Projekten hervor. Das sogenannte „Quadrat des Teufels“ zeigt, welche Dimensionen justiert werden können und wie diese sich gegenseitig beeinflussen.

Im hybriden Modell vereinen sich agile und klassische Methoden. Während die Methoden sich in anderen Anwendungsbereichen ergänzen, unterscheidet sich die Vorgehensweise für die Aufwandsschätzung bei agil und sequentiell durchgeführten Projekten deutlich. Die bedeutendsten Unterschiede bestehen bzgl. des Zeitpunktes des Einsatzes einer ausgewählten Schätzmethode, der Verantwortlichkeit für die Schätzung, der zur Verfügung stehenden Informationen, der technischen Rahmenbedingungen und der verfolgten Zielstellung. Bei

[8] Vgl. Klaus (2012), S. 8.

[9] Vgl. Sneed (2007a), S. 55.

agilen Methoden ist die Schätzung ein integraler Bestandteil der auszuführenden Tätigkeit und wird zu Beginn einer jeden Iteration durchgeführt[10]. Hier zeigt sich die Diskrepanz zu Festpreisprojekten, bei denen die Aufwandsschätzung bereits zu Beginn innerhalb der Projektinitiierung vorgenommen wird, um ein initiales Angebot abgeben zu können.

Eine durch Schmietendorf durchgeführte Expertenbefragung im Jahr 2008 zeigte, dass Entwickler nach einem agilen Vorgehen eine Anpassung klassischer Methoden zur Aufwandsschätzung in den Vordergrund stellen. Allerdings hat sich auch gezeigt, dass die grundlegenden Verfahren zur Aufwandsschätzung, wie das COCOMO- oder das Function-Point-Verfahren bei einem agilen Vorgehen sich weder für die Schätzung noch zur Erfassung des *Load Factors*[11] eignen[12]. Klassische Ansätze wie das COCOMO- oder das Function-Point-Verfahren berücksichtigen primär die folgenden Aspekte[13]:

- Funktionaler Umfang (Function Points)
- Implementierungstechnischer Umfang (COCOMO)
- Komplexität der Anforderungen
- Technologische Einflüsse
- Art der verwendeten Entwicklungsumgebung
- Aufwandsermittlung in Bezug auf korrespondierende Erfahrungen

Den klassischen Verfahren fehlen allerdings Aspekte, die der Komplexität eines IT-Systems gerecht werden. Mit dem Aufkommen der objektorientierten Entwicklung, der verstärkten Wiederverwendung von Komponenten, dem Einsatz von Frameworks Dritter und der Entwicklung serviceorientierter Architekturen werden folgende Komplexitätskriterien in der Aufwandsschätzung berücksichtigt:

- Prozesskomplexität
- Datenkomplexität
- Servicekomplexität
- Technologische Komplexität

Diese eigentlich für die Aufwandsschätzung von SOA-Projekten konzipierten Kriterien können auf alle Arten von Softwareprojekten übertragen werden. Ein Verfahren, das diese zusätzlichen Komplexitätskriterien berücksichtigt, ist das COSMIC-Verfahren. Es wurde 1998 auf Basis der Function-Point-Methode entwickelt und bietet in Anlehnung an den ISO-Standard 14143 Verfahren, Prinzipien, Regeln und einen Prozess zur Auszählung bzw. Messung der fachlich determinierten Größen einer Softwarekomponente.

---

[10] Vgl. Schmietendorf und Dumke (2009), S. 54.

[11] Der Load Factor beschreibt das Verhältnis einer ausgewiesenen Ressource (Mensch, Maschine oder System) zur Zeit, die die Ressource wirklich zur Verfügung steht.

[12] Vgl. Schmietendorf und Dumke (2009), S. 55, 57.

[13] Vgl. Schmietendorf und Dumke (2010), S. 77.

Aber „Die Methode basierte auf einem sehr wackeligen und individuell auslegbaren Zählungsprozess sowie auf einer fragwürdigen Umsetzungstabelle, um die gezählten Function-Points in Personenmonate umzusetzen.“[14] Außerdem kann die Function-Point-Methode nur angewendet werden, wenn Ein- und Ausgaben vollständig bekannt sind. Dies ist im Vorfeld aber nur selten der Fall.[15]

Mit der Objektorientierung und der verstärkten Betonung von Wiederverwendung hat der Function-Point seine Bedeutung als Größenmaß verloren. Bei der Objektorientierung steht die Datenseite und weniger die funktionale Betrachtung von Ein- und Ausgabe im Vordergrund. Es bieten sich zwei neuere Verfahren für die Aufwandsschätzung an, die bereits in einem frühen Stadium des Projekts (in der Analyse- und Designphase) ein aussagekräftigeres Maß für die Systemgröße bieten[16]:

- **Object-Point:** Das Object-Point-Verfahren geht von dem Objektmodell bzw. den korrespondierenden UML-Diagrammen aus. Object-Points werden aus den Größen Anzahl Klassen, Anzahl Schnittstellen, Anzahl Klassenbeziehungen, Anzahl Methoden und Anzahl Attribute abgeleitet. Alle Faktoren gehen aus dem UML-Diagramm hervor und eignen sich für eine automatische Zählung. Dafür sollte das Objektmodell allerdings möglichst vollständig dokumentiert sein.
- **Use-Case-Point:** Im Use-Case-Point-Verfahren werden ausschließlich die Use-Case-Diagramme zur Bestimmung der Maßgröße herangezogen. Es eignet sich besonders, um den Aufwand für neue Systeme ohne Erfahrungshorizont zu schätzen. Bei der Ableitung aus den Use-Case-Diagrammen wird jeder Systemakteuer bzw. jede Systemschnittstelle als einfach, mittelkomplex oder komplex eingestuft. Gleichzeitig werden die Anwendungsfälle klassifiziert und gewichtet.

Das Use-Case-Point-Verfahren kann früher in einem Projekt zur Bestimmung des Aufwands herangezogen werden, da die Geschäftsvorfälle bereits früh modelliert werden. Ebenfalls wird die Komplexität von Systemen betrachtet und findet durch passende Faktoren ausreichend Gewichtung in der Aufwandsschätzung. Allerdings fehlen bei der Use-Case-Point- und der Object-Point-Methode die vielen Erfahrungsdatenbanken, die mit dem Function-Point-Verfahren in den vergangenen Jahren aufgebaut wurden.

Diese umfangreichen Datensammlungen lassen sich bei der International Software Benchmarking Standards Group (ISBSG)[17] abrufen und für ein Benchmarking einsetzen. Die Nutzung einer Benchmark kann für die Vor- und Nachkalkulation von Projekten angewendet werden, um die eigenen Daten zu verifizieren. Häufig wird die Vorkalkulation bzw. die Nachkalkulation in Form einer Revision nur halbherzig oder gar nicht durchgeführt. Dabei sind Erfahrungswerte hilfreich für die verlässliche Schätzung zukünftiger Projekte.

---

[14] Sneed (2007b), S. 73.

[15] Vgl. ebd., S. 75, 77.

[16] Vgl. Peischl und Wuksch (2013), S. 54; vgl. Sneed (2007b), S. 77 f.

[17] www.isbsg.org.

Abschließend kann keine Empfehlung für die richtige Methode zur Aufwandsschätzung gegeben werden. Die Auswahl der Methode hängt von den vorliegenden Informationen des Auftraggebers ab. Lassen sich daraus Anwendungsfälle ableiten, bietet sich das Use-Case-Point-Verfahren an. Insbesondere wenn es sich um innovative Projekte ohne Vergleichsbasis handelt, ist es schwierig, den Aufwand korrekt zu schätzen.

5.2.3 Nutzenanalyse

Allgemein kann der Nutzen eines Softwareprojekts in die Bereiche Kostenersparnis, Produktivitätsverbesserung und strategische Wettbewerbsvorteile aufgegliedert werden[18]. Der Gesamtnutzen eines Projekts setzt sich aus Grund- und Zusatznutzen zusammen. Im Rahmen eines Softwareprojekts steht der Zusatznutzen nicht direkt mit dem Grundnutzen in Beziehung. Als Beispiel kann hier der Erfahrungsgewinn mit einer bestimmten Technologie genannt werden. Weiterhin kann der Nutzen in wirtschaftlichen und funktionalen Nutzen unterschieden werden. Im hybriden Modell findet die Priorisierung der Funktionalität anhand des wirtschaftlichen Nutzens statt, indem der Auftraggeber messbare Geschäftsziele vorgibt. Dadurch soll vermieden werden, dass zu viel Arbeit in unnötige Funktionen gesteckt wird, die am Ende das Budget belasten, den wirtschaftlichen Nutzen des Projekts aber nicht steigern.

Typische Projektarten, die sich aus den Geschäftszielen ableiten sind im Folgenden aufgelistet:[19]

- **Rationalisierungsprojekte** mit den Nutzenfaktoren Personaleinsparungen, Produktivitätserhöhung oder Einsparung von monetären Aufwänden (Material, Zinsen, Skonto). Bei dieser Form von Projekt lässt sich der Nutzen monetär (z. B. in Form eingesparter Personalkosten) quantifizieren.
- **Der Aufbau von Informationssystemen** mit den Nutzenfaktoren Verkürzung der Durchlaufzeiten, erhöhter Transparenz im Geschäftsprozess oder besserer Kundenservice. Der Grundnutzen ist bei dieser Form von Projekten nicht oder kaum monetär zu quantifizieren, denn die Effekte ergeben sich meist über den Zusatznutzen.

Eine mögliche Methode zur Nutzenbewertung ist die von Blumberg et al., in der umfangreiche Anforderungen zur Nutzenbewertung berücksichtigt sind. Dazu gehören die Monetarisierung des Nutzens, die Berücksichtigung der Risiken sowie die von Abhängigkeiten bei gleichzeitiger Sicherstellung der Praktikabilität der Methode.[20]

Die bisher in Forschung und Praxis genutzten Methoden, wie das Scoring-Modell, das WARS-Modell, die Wirkungskette, die Balanced-Scorecard und das SMART-Modell, arbeiten mit Punkten, die bestimmten Kriterien zugeordnet werden. Bei der Monetarisie-

[18] Vgl. Henrich (2002), S. 163.

[19] Vgl. ebd., S. 165 ff.

[20] Vgl. Blumberg et al. (2012), S. 57.

rung dieser Punkte finden häufig subjektive Bewertungen statt, die das Ergebnis verzerren. Andere Modelle betrachten das Risiko nicht ausreichend.[21]

Die BeneFIT-Methode soll die Schwächen der bisherigen Modelle ausgleichen. Dazu werden vier Komponenten betrachtet[22]:

- **Quantifizierung des Nutzens:** Die Stakeholder definieren Performance-Steigerungen und Einsparpotenziale, die durch das Projekt erwirkt werden wie z. B. Verringerung von Schulungszeiten, Automatisierung von Prozessen und Zusammenführung von Systemen. Der Nutzen wird also an der Steigerung des Unternehmenswerts gemessen, denn der Nutzen führt direkt oder indirekt zu Einsparungen von bisher auftretenden Zahlungen. Da Prognosen Unsicherheiten bergen, wird kein einzelner Wert geschätzt, sondern ein Wahrscheinlichkeitsintervall von 80 % auf Basis einer Normalverteilung berücksichtigt, in dem der Nutzen sich monetär wiederfindet
- **Erfassung der Risiken:** Zu dem Erwartungswert eines Nutzens wird anschließend das Risiko ermittelt. Der Nutzen eines Projekts hängt vom Marktrisiko und weiteren Einflussfaktoren ab. Damit kommt es zu einer Abweichung vom ursprünglichen Wert. Der erwartete Gesamtprojektwert ergibt sich schließlich aus der Aggregation der verschiedenen Erwartungswerte und den deterministisch ermittelten Auszahlungen des Projekts.
- **Berücksichtigung von Abhängigkeiten:** Nutzen können miteinander korrelieren. Ist dies der Fall, so können auch Risiken korrelieren. Daher werden die Abhängigkeiten erfasst und nach Eintrittswahrscheinlichkeit gewichtet. Daraus ergibt sich ein monetärer Risikoabschlag. Dieser risikoadjustierte Gesamtprojektwert ist die zentrale Kennzahl für die Betrachtung des Projekts nach dem erfolgreichen Abschluss, kann aber bereits zur Projektsteuerung eingesetzt werden.
- **Projektsteuerung:** Die Projektsteuerung basiert auf einem risikoadjustierten Restprojektwert. Wird im Verlauf des Projekts eine Verschlechterung der Zahlungsströme festgestellt, findet eine Neubewertung des Projekts statt, die ggf. einen sofortigen Abbruch nahelegt. Allerdings bleibt bei dieser Steuerung der *Sunk-Cost-Effekt* unberücksichtigt. Der Sunk-Cost-Effekt beschreibt die Theorie, dass Menschen umso länger an einer Handlungsalternative (beispielsweise einem Investitionsprojekt oder einer Marketingkampagne) festhalten, je mehr Geld, Zeit oder Arbeitskraft in der Vergangenheit investiert worden ist. Laut Gleißner ist der Sunk-Cost-Effekt umso größer, je höher der Investitionsanteil an den Gesamtkosten ist (Geld), je weiter der Investitionszeitraum oder die Handlung fortgeschritten ist (Zeit), je seltener Menschen eine vergleichbare Si-

[21] Vgl. Blumberg et al. (2012), S. 57.

[22] Die Autoren sprechen im begleitenden Artikel von drei Komponenten. In der Methodik sind die Quantifizierung des Nutzens und die Erfassung der Risiken separate Aktivitäten, die zum Erfassungszeitpunkt noch nicht in Relation stehen. Es ist daher sinnvoll von vier strukturgebenden Komponenten zu sprechen; vgl. Blumberg et al. (2012), S. 58 f.

tuation erlebt haben und je unklarer die Situationen und Handlungsalternativen sind.[23] Laut Eisenführ und Weber sind von dieser Verzerrung insbesondere Entscheidungen über die Fortführung von Projekten betroffen[24].

Im Vergleich zu den anderen genannten Scoring Modellen ermöglicht die BeneFIT-Methode eine neutralere Bewertung des Nutzens. Ihre Anwendung ist jedoch aufwändig.

5.2.4 Prototyping und Feedbackmechanismen

Durch das Prototyping wird in frühen Phasen sichergestellt, dass die Kundenanforderungen mit dem Projekt umgesetzt werden. Continuous Integration und Continuous Delivery unterstützen (siehe auch Kap. 5, Abschn. 5.2.7). das Entwicklungsteam bei der Erstellung von Prototypen, weil sie regelmäßig ausführbaren Code produzieren, der durch den Kunden verifiziert wird.

Prototyping unterstützt das schnelle Feedback, mit dem es möglich ist, zeitnah auf die Bedürfnisse oder Veränderungen seitens der Kunden oder auch auf Qualitätsprobleme zu reagieren[25]. Schnelles Feedback lässt sich in sequentielle und iterative Modelle integrieren, sodass diese Methode für das hybride Modell geeignet ist.

5.2.5 Test-Driven Development

Beim Testen wird der Nachweis erbracht, dass die getestete Funktionalität wiederholt einwandfrei funktioniert. Je sicherheitskritischer eine Funktion oder Komponente ist, umso sorgfältiger sollte die Softwareentwicklung und Qualitätssicherung erfolgen.

Um eine hohe Qualität zu erreichen, empfehlen Tom und Mary Poppendieck Test Driven Development (TDD). Bei dieser Methode werden Softwaresysteme so oft wie möglich automatisch, z. B. jede Stunde, getestet. Wenn ein Fehler festgestellt wurde, dann wird kein weiterer Code hinzugefügt, bis das Problem gelöst wurde.[26] TDD vermeidet so, dass erst am Ende der Implementierung die Testfälle unter Zeitdruck und mit einer unzureichenden Abdeckung erstellt werden. Eine weitere Forderung des TDD im Hinblick auf den Ten-Minute-Build ist, dass die komplette Test Suite für ein Modul bzw. eine Funktionalität in wenigen Minuten zu einem Ergebnis führt[27]. Der Entwickler soll nicht lange auf das Feedback seiner Tests nach einer Integration warten.

---

[23] Vgl. Blumberg et al. (2012), S. 58 f.

[24] Vgl. Eisenführ und Weber (2003), S. 371.

[25] Vgl. Ekssir (2013), S. 11.

[26] Vgl. Poppendieck (2007), S. 27.

[27] Vgl. Beck (2003); vgl. Stober und Hansmann (2010).

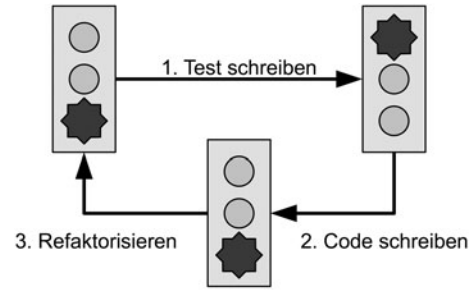

Abb. 5.4 Ablauf des TDD-Zyklus. (Eigene Darstellung in Anlehnung an Gärtner (2013), S. 114)

Sobald das Design geändert wird, erhalten Entwickler umgehend Feedback durch die vorher geschriebenen Unit-Tests[28]. Ein testgetriebener Entwicklungszyklus besteht aus drei Schritten. Er beginnt mit dem Schreiben eines Tests, der zunächst noch fehlschlägt, da die entsprechende Funktionalität noch nicht implementiert wurde. Danach wird gerade soviel Code geschrieben, dass der Test erfolgreich durchlaufen wird. Abgeschlossen wird der Zyklus durch ein Refactoring bei dem der Code aufgeräumt und vereinfacht wird. Der erstellte Test verhindert dabei, dass Fehler in die erstellte Funktionalität eingebaut werden.[29] Abbildung 5.4 stellt diesen Ablauf schematisch dar:

Forschungen von Nokia Solutions Networks zeigen, dass Entwicklungen nach TDD ca. 15 % mehr Zeit benötigen, die Qualität des Codes aber deutlich besser und die Anzahl der Defects um bis zu 40 % niedriger ist[30].

TDD kann allerdings nicht in jeder Situation angewendet werden. Bei Fehlern, die schwierig zu reproduzieren sind und die eine komplexe Umgebung für den Test benötigen, lohnt sich die Automatisierung der Tests nicht. Außerdem kann das TDD in der Anfangsphase und bei ungeübten Programmierern die Entwicklung um den Faktor zehn verlangsamen[31]. Bei bestimmten Designvorgaben, z. B. wenn das Design für einen Test zu öffnen ist, kann TDD ebenfalls nicht verwendet werden[32]. Unter der Öffnung des Designs verstehen Softwareentwickler die Notwendigkeit, die Sichtbarkeit von Variablen oder Methoden durch Veränderung der Zugriffsrechte zu modifizieren. Die Zugriffsrechte werden dabei erhöht, sodass bspw. eine ursprünglich als privat deklarierte Variable oder Methode für den Test mindestens protected oder sogar public ist. Dies ist in sicherheitskritischen Umgebungen nicht gewünscht.

[28] Vgl. Cockburn (2003), S. 225.
[29] Vgl. Gärtner (2013), S. 114.
[30] Vgl. Marchenko et al. (2009), S. 18 f.
[31] Vgl. ebd., S. 20.
[32] Vgl. ebd.

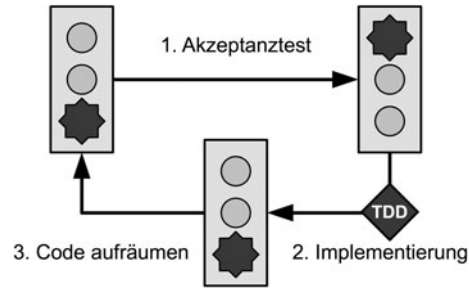

Abb. 5.5 Schematischer Ablauf von ATDD. (Eigene Darstellung in Anlehnung an Gärtner (2013), S. 114)

5.2.6 Acceptance Test-driven Development

Bei Acceptance Test-driven Development (ATDD) handelt es sich um eine Maßnahme, mit der Anforderungen möglichst korrekt im Code umgesetzt werden sollen.[33]

„Ein Akzeptanztest überprüft ein System auf seine funktionalen Eigenschaften. Er beantwortet die Frage: Verhält sich die Anwendung fachlich wie erwartet?“[34] Diese Validierung, die aus der Spezifikation abgeleitet wird oder sogar die Spezifikation bilden kann, wird vor der Erstellung des Codes geschrieben.[35] Auch Coldewey empfiehlt Anforderungen direkt als Akzeptanztests statt in Anforderungsdokumenten aufzuschreiben und zwar bevor das System die Funktionalität unterstützt. Üblicherweise werden zuerst nur die erwarteten Ergebnisse als Akzeptanztests angelegt und dann mögliche Fehler- und Problemfälle nachgeliefert.[36] Die Abb. 5.5 zeigt den schematischen Ablauf bei ATDD.

Automatisierte Akzeptanztests lassen sich auch in einen Continuous Build integrieren (siehe dazu auch Kap. 5, Abschn. 5.2.7). Damit wird nach jedem Build automatisch der Akzeptanztest ausgeführt, um sicherzustellen dass die Software den fachlichen Anforderungen entspricht.[37] Allerdings können automatisierte Akzeptanztests aufmerksame Tester nicht ersetzen, denn es gibt Bereiche, die sich kaum automatisiert testen lassen, wie z. B. die Bedienbarkeit eines Systems.[38]

Zusammen mit Test Driven Development (siehe Kap. 5, Abschn. 5.2.5) lässt sich durch ATDD ein hoher Grad an Testautomatisierung erreichen. Test Driven Development stellt sicher, dass der Code keine Fehler enthält und ATDD unterstütz die Erfüllung der fachlichen Anforderungen.[39]

---

[33] Vgl. Gärtner (2013), S. 115.
[34] Bepple (2011), S. 97.
[35] Vgl. Poppendieck (2010), S. 73.
[36] Vgl. Coldewey (2008).
[37] Vgl. Bepple (2011), S. 99.
[38] Vgl. ebd., S. 99.
[39] Vgl. Gärtner (2013), S. 116.

5.2.7 Continuous Integration

Verglichen mit dem einmaligen Integrationsprozess am Ende eines Softwareprojektes verbessert Continous Integration (CI) laut Wiest die Produktqualität[40]. Es treten weniger Fehler auf, sofern entsprechende Maßnahmen bzgl. einer ausreichenden Testabdeckung, statischer Codeanalyse und eines vollständig aufgebauten CI-Prozesses durchgeführt werden.

Durch CI wird das Fehlerrisiko am Ende eines Softwareprojekts minimiert, da diese schon im Vorfeld innerhalb der Iterationen frühzeitig entdeckt werden. Metriken zur Testabdeckung (Test Coverage) treffen eine Aussage über die Qualität der Tests und der Fehlertoleranz der Software. Eine niedrige Test Coverage bedeutet, dass Software auf viele mögliche Standardfehler nicht korrekt reagiert und es zu Folgefehlern in der Verarbeitung kommt.

5.2.8 Continuous Delivery

Unter Continuous Delivery (CD) wird die frühe, häufige und vollautomatische Auslieferung und Installation von Software-Artefakten beim Kunden verstanden[41]. Nachdem Komponenten, die zu einem Release gebündelt werden, alle automatischen Integrationstests bestanden haben, wird das Deployment vorbereitet. CD setzt dafür ein hohes Maß an Automatisierung und die Definition konkreter Metriken für die Qualitätssicherung der Software voraus. Durch diesen Prozess sollen manuelle Releases eliminiert werden[42]. Nach Cockburn wird CD als das Ideal der agilen Softwareentwicklung betrachtet. Durch frühe und häufige Lieferungen erhalten die Entwickler ein frühes Feedback von den Kunden[43]. Cockburns Einschätzung nach sollten die Lieferungen nicht zu häufig stattfinden, sondern in ca. dreimonatigen Zyklen erfolgen, um den Kunden nicht zu überfordern[44].

CD eignet sich nicht für alle Softwareentwicklungsprojekte. Z. B. bei Software in sicherheitskritischen Bereichen empfiehlt sich der Einsatz von CD nicht, weil umfangreiche Abnahmetests und Audits vor Inbetriebnahme der Software notwendig sind. Im Bereich der mobilen Anwendungsentwicklung oder der Web-Entwicklung haben sich CI und CD jedoch bereits etabliert.

Die Kombination von CD mit dem ITIL™ Change Management ist widersprüchlich, denn das Change Management sieht die Prüfung jedes RFCs durch den Change Manager vor (siehe 2.11.2). Da das Deployment bei CD automatisiert und zeitnah nach der Erfüllung aller Integrationstests erfolgt, bleibt dem Change Manager somit keine Zeit für eine Prüfung.

---

[40] Vgl. Wiest (2011), S. 24 ff.

[41] Vgl. Kamann und Wendt (2012), S. 58 f.

[42] Vgl. ebd., S. 58 f.

[43] Vgl. Cockburn (2003), S. 287.

[44] Vgl. ebd.

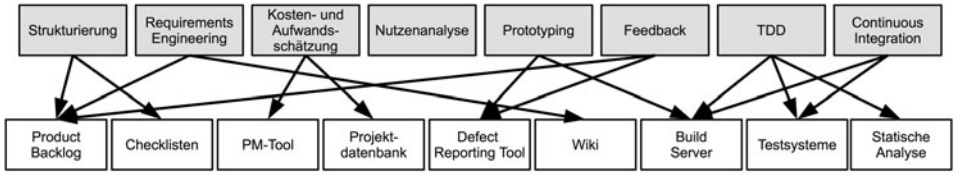

Abb. 5.6 Zuordnung der Methoden und Werkzeuge. (Eigene Darstellung)

Ein Unternehmen hat daher zu entscheiden, ob die formale Prüfung durch den Change Manager oder CD erfolgt. Das hybride Modell unterstützt beide Vorgehensweisen.

5.3 Techniken und Werkzeuge

Die nachfolgend vorgestellten Werkzeuge und Techniken unterstützen die Projektbeteiligten bei der Arbeit. In Abb. 5.6 gezeigten Werkzeugen und Methoden lassen sich wie folgt beschreiben:

- **Product Backlog:** Bei agilen Vorgehensweisen kommen Änderungen häufig vor. Um z. B. für kritische Systemen im Rahmen der Qualitätssicherung und der Risikosteuerung alle Anforderungen über die Implementierung bis hin zu den Testergebnissen nachverfolgbar zu halten, sollte diese Nachverfolgbarkeit werkzeuggestützt erfolgen. Die Granularität der Dokumentation ist so gering wie möglich zu halten, um den bürokratischen Aufwand im hybriden Modell zu minimieren.
- **Checklisten:** Checklisten beinhalten die Tätigkeiten, die ein Projekt-Team bereits durchführt. Sie dienen einem Gutachter oder Auditor als Beleg für die Tätigkeiten und Prozesse der Organisation im Rahmen der Qualitätssicherung[45].
- **Projektmanagement-Tool:** Ein elektronisches Tool für das Projektmanagement unterstützt die Rückverfolgbarkeit. Dadurch werden Konformitätsanforderungen der Qualitätssicherung (insbesondere der Prozessqualität) sichergestellt[46].
- **Projektdatenbank:** Eine Projektdatenbank unterstützt die Vor- und Nachkalkulation von Projekten. Sie ist für den Aufbau von historischen Daten erforderlich, um Aufwands- und Kostenschätzungen durchzuführen. Liegt einer Organisation keine eigene Datenbasis als Benchmark vor, so sind öffentlich zugängliche Datenbanken mit Projektdaten im Internet verfügbar. Allerdings lassen sich die dort vorhandenen Daten, Erfahrungen und hinterlegten Aufwände nicht auf jedes Projekt und jede Organisation übertragen.[47]

[45] Vgl. Cohn (2010), S. 430.
[46] Vgl. ebd.
[47] Vgl. Peischl und Wuksch (2013), S. 50 f.

- **Defect Reporting Tool:** Das Defect Reporting Tool dient der Nachverfolgung von Fehlern in dem erstellten Produkt. Damit wird die Produktqualität sichergestellt und Konformitätsanforderungen zur Sicherung der Prozessqualität erfüllt.[48]
- **Wiki:** Das Wort Wiki stammt aus der hawaiianischen Sprache und bedeutet übersetzt „schnell". Wikis dienen also der schnellen Dokumentation. Häufig degenerieren Wikis aber zu einem unorganisierten Sammelplatz für verschiedene Themen[49] oder es werden gleiche Themen mit unterschiedlichen oder falschen Erkenntnissen erfasst. Bei der Einführung eines Wikis in der Organisation empfiehlt sich darum der Einsatz eines verantwortlichen Redakteurs, der das Wiki sinnvoll steuert.
- **Testsysteme:** Testsysteme sind notwendig, um die Prototypen auf einem System zu demonstrieren. Die Systeme sollten dabei weitestgehend den zukünftigen Systemen im Einsatz beim Auftraggeber entsprechen.
- **Build Server:** Build Server oder CI-Server dienen dazu, die Software automatisch zu bauen. Sie sind die Basis für die CI. Erst wenn der Bau der Software vollständig automatisiert ist, kann mit der Continuous Integration begonnen werden.
- **Statische Analyse:** Review- und Inspektionstechniken decken Anomalien im Code auf[50]. In Verbindung mit einem CI-System weist die statische Code-Analyse den Entwickler automatisch auf Anomalien und problematische Stellen im Code hin.

5.4 Unterstützende Prozesse

In Kap. 3, Abschn. 3.5.8 wurde die Bedeutung unterstützender Prozesse in der Softwareentwicklung herausgestellt. Im hybriden Modell werden die aus den agilen und phasenorientierten Modellen genannten Prozesse übernommen.

Auch agile Vorgehen wie Scrum berücksichtigen einen KVP und fördern die kontinuierliche Qualitätssicherung. Dabei empfiehlt sich die Auslagerung umfangreicher Funktionen in separate Prozesse.

5.4.1 Kontinuierlicher Verbesserungsprozess

Entsprechend den meisten Normen und Zertifizierungen in der Qualitätssicherung kann eine Organisation einem eigenen Qualitätsmanagementsystem folgen und somit selbst Anpassungen vornehmen[51].

[48] Vgl. Korhonen (2009); vgl. Lange et al. (2013), S. 26.
[49] Vgl. Wiegand (2013), S. 3.
[50] Vgl. Liggesmeyer (2009), S. 16.
[51] Vgl. Cohn (2010), S. 431.

Zu den bekanntesten Process Improvement Modellen gehören CMMI, Six Sigma, Control Objectives for Information and Related Technology (COBIT), CMM, Kaizen und Software Process Improvement and Capability Determination (SPICE). SPICE ist die europäische Alternative zu CMM und CMMI und bietet deutlich mehr Gestaltungs- und Anpassungsmöglichkeiten an die Unternehmenssituation. Eine Umfrage von Rechberger und Nissl zum Einsatz von Process Improvement Modellen im deutschsprachigen Raum hat ergeben, dass der Hauptvorteil in der Verbesserung der Qualität der Prozesse liegt. Weiterhin geben die Befragten an, dass diese Modelle zu einer erhöhten Transparenz, zum Einsatz von Best Practices und zur Kostenersparnis auf lange Sicht führen. Für einige Unternehmen ist der Nachweis der Anwendung einer der o. g. Methoden erforderlich, um zur Teilnahme an Ausschreibungen zugelassen zu werden. Allerdings haben die Modelle bei Entwicklern häufig einen schlechten Ruf, denn sie benötigen einen hohen zeitlichen Aufwand und stören das Tagesgeschäft bei der Einführung im Unternehmen.[52]

5.4.2 Qualitätssicherung

Qualitätssicherung erstreckt sich über den Entwicklungsprozess bis hin zur Evolutionsphase. Qualitätsziele werden darum nicht nur für die Abnahme, sondern auch für darauf folgende Wartungs- und Evolutionsphasen definiert. Sie sind bereits für die frühen Phasen der Entwicklung (Analyse und Design) festzulegen. Zur Definition von Zielen lassen sich Qualitätsdimensionen unterscheiden in funktionale Korrektheit, Performanz-Aspekte, Benutzbarkeit, Sicherheit, Wartbarkeit und Veränderbarkeit/Anpassbarkeit. Um die Zielerreichung für diese Dimensionen überprüfbar zu machen, werden Qualitätssicherungsziele quantitativ bewertet. In der Literatur werden für Qualitätssicherungsmaßnahmen häufig zwei Verfahrensziele aufgezeigt: der Beweis von Korrektheit bestimmter Funktionalität oder die Vermittlung von Vertrauen[53]. Zur Prüfung der Korrektheit reichen formale Spezifikation und automatisierte Tests häufig nicht aus, um die funktionale und formale Korrektheit zu beweisen, da auch die ausgeführten Tests fehlerhaft sein können. „[Es gilt,] ... dass durch Testen zwar die Anwesenheit von Fehlern, nicht aber deren Abwesenheit gezeigt werden kann.“[54]

Korhonen hat in einer Studie gezeigt, dass das Management von Defects nach agilen Prinzipien verbessert werden kann. Als Probanden dienten drei Organisationen, die sich nach Größe der Organisation, Vorgehensmodell und Art der Software unterschieden. Zwei Organisationen setzten auf agile Methoden, die dritte setzte das Wasserfallmodell ein, um ihr Bestandsprodukt zu pflegen[55].

---

[52] Vgl. Rechberger et al. (2007), S. 51, 53–55.

[53] Vgl. Mittermeir (2004), S. 27 f.

[54] Liggesmeyer (2009), S. 16.

[55] Vgl. Korhonen (2009), S. 76.

Die Studie kommt zu dem Ergebnis, dass sich der Einsatz eines Defect Reporting Tools empfiehlt, in dem alle Fehler einer Software erfasst werden. Dieses Tool ist vergleichbar mit einem Backlog nach Scrum. Ein Entwickler entscheidet, welcher Fehler in der nächsten Iteration behoben wird. Zeitgleich ziehen die Qualitätsmanager bzw. der Projektleiter aus den automatisch erhobenen Metriken Rückschlüsse auf die Qualität des Produkts und einzelner Komponenten. Die Metriken zur Messung der Produktqualität sind unabhängig vom Vorgehensmodell. In allen drei untersuchten Organisationen war die Anzahl der noch zu behebenden Fehler das Hauptkriterium, um eine Aussage über die Produktqualität zu treffen. Die Messung erfolgt allerdings nicht nach jedem offiziellen Release, sondern nach jeder Iteration. So wird der Fortschritt schneller kommuniziert und Kunde und Entwickler erhalten frühzeitig Feedback[56].

Für die vorgestellte Methodik gelten Regeln, die den Qualitätssicherungsprozess unterstützen[57]:

- Vor dem Einsatz eines Defect Reporting Tools wird ein Leitfaden erstellt, wann und welche Arten von Fehlern berichtet werden. Findet z. B. ein Entwickler innerhalb einer Iteration einen Fehler, so kann er diesen direkt beheben, ohne diesen zu berichten.
- Es gelten klare Richtlinien für die Fehlerbehebung und die Implementierung von neuen Funktionen. Kleinere Fehlerbehebungen können bspw. in die nächste Iteration verschoben werden. Ggf. wird innerhalb einer Iteration ein Zeitraum eingeräumt, der nur der Fehlerbehebung dient.
- Bei der Erhebung von Metriken zur Produktqualität werden Metriken aus phasenorientierten Modellen genutzt. Dazu gehören die Gesamtanzahl der offenen Fehler, die Anzahl offener Fehler je nach Komponente, der geschätzte Aufwand zur Fehlerbehebung, etc.

In der Softwareentwicklung haben sich in den letzten zehn Jahren die zwei Modelle CMMI und SPICE etabliert, die Unternehmen bei der Verbesserung ihrer Prozesse unterstützen. Lami und Falcini haben in einer Untersuchung gezeigt, dass agile Methoden mit den Modellen zur Prozessqualitätsmessung nicht in Konflikt stehen und keinen streng formalisierten und dokumentenbasierten Ansatz bei der Prozessdokumentation benötigen. SPICE verlangt nur die Dokumentation der notwendigen Prozesse, die für ein Projekt charakteristisch sind und die Ziele der Organisation unterstützen. Allerdings geben die Autoren auch zu bedenken, dass mit rein agilen Ansätzen maximal der SPICE Maturity Level 3 erreicht werden kann[58]. Begründet liegt dies an den Anforderungen zur Erfüllung des Process Reference Models. Agile Modelle können diese nicht vollständig erfüllen, da die vorgestellten Ansätze keine Phasenmodelle und Prozesse unterstützen, sondern in erster Linie Methoden sind. Das hybride Modell ermöglicht höhere Level, indem es die Praktiken in ein Prozessmodell integriert, das die agilen Methoden unterstützt, aber nicht einschränkt.

---

[56] Vgl. Korhonen (2009), S. 77–79.

[57] Vgl. ebd., S. 80.

[58] Vgl. ebd., S. 428 ff.; vgl. Lami und Falcini (2009), S. 130 ff.

5.4.3 Projektmanagement

Im hybriden Modell entspricht das Projektmanagement der klassischen Definition. Hauptsächlich ist es für die Planung und Kontrolle des Projektfortschritts und die Kommunikation mit dem Auftraggeber auf vertraglicher Ebene zuständig. Zu den Aufgaben des Projektleiters gehört es, in Zusammenarbeit mit der Unternehmensführung des Auftragnehmers und des Auftraggebers das Risikomanagement zu übernehmen. Das Projektmanagement koordiniert aber auch die Softwareerstellung und die Qualitätssicherung.[59] Inwiefern das Projektmanagement intern nach agilem oder klassischem Vorgehen organisiert ist, hängt von der Projektgröße und den Anforderungen des Auftraggebers ab.

5.4.4 Risikomanagement

Für das Risikomanagement im hybriden Modell gelten die in Kap. 3, Abschn. 3.5.8 vorgestellten Bedingungen. Aufgabe des Risikomanagements ist es, kritische Aktivitäten im Projektverlauf rechtzeitig zu erkennen und Gegenmaßnahmen zu planen bzw. zu ergreifen. Die häufigsten Risikofaktoren, die auch zu einem Scheitern von Projekten führen, sind dabei Termindruck, Ressourcenengpass oder -ausfall, komplexe Abstimmungen und ständige Modifikation der Anforderungen.

Durch die zusätzliche Einbindung von agilen Praktiken in das hybride Modell können durch geeignete Feedbackmechanismen diese Risiken erheblich reduziert werden. Zu den Feedbackmechanismen gehören[60]:

- **Daily Stand-Up Meeting:** Das Daily Stand-Up Meeting findet zwischen Projektleiter und den einzelnen Entwicklungsteams statt. Die Besprechungen dauern dabei nur wenige Minuten in denen jeder Softwareentwickler berichtet, woran er gerade arbeitet, wie lange er die Dauer schätzt und welche Probleme er im Vorfeld erwartet. Auf Basis dieses täglichen Feedbacks kann der Projektleiter seine interne Risikoanalyse erstellen.
- **Iterations-Planung:** In einer strukturierten Iterations-Planung wird zunächst die Dauer aller Iterationen festgelegt und der Funktionsumfang der Prototypen am Ende jeder Iteration definiert. Dadurch erhält der Kunde einen Überblick über die schrittweise Entwicklung des Produktes. Änderungen sind nach jeder Iteration möglich und in der Aufwandskalkulation zu berücksichtigen.
- **Review:** Das Review am Ende jeder Iteration, das in Zusammenarbeit mit dem Auftraggeber durchgeführt wird, ist die wichtigste Tätigkeit zur Risikovermeidung. Vor dem Review hat der Kunde ausreichend Zeit, einen Prototypen mit dem gewünschten

[59] Vgl. Hansen und Neumann (2005), S. 277, 282.

[60] Vgl. Schmietendorf (2013b), S. 68.

Funktionsumfang zu testen. Im Review teilt er dem Auftragnehmer mit, ob seine Anforderungen korrekt verstanden worden sind und welche Änderungen er wünscht oder welche kritischen Fehler im Test gefunden wurden. Somit können Abstimmungsprobleme und falsch verstandene Anforderungen im Vorfeld geklärt werden. Nachbesserungen finden so nicht erst zum Ende des Projekts oder innerhalb der Gewährleistungsphase statt. Durch das Review sind sowohl Auftraggeber, als auch Auftragnehmer mit verantwortlich für die Risikoabschätzung. Der Projektleiter übernimmt eine koordinierende Rolle, bei der alle Informationen zum Risikomanagement zusammenkommen.

Die Kombination von klassischen und agilen Methoden optimiert das Risikomanagement. Durch das frühzeitige Feedback des Auftraggebers und die Aufdeckung von Problemen im Entwicklungsprozess während interner Besprechungen können Risiken früher aufgedeckt werden.

5.5 Skalierbarkeit und Flexibilität

Anhänger verschiedener Vorgehensweisen behaupten immer wieder, dass ihr Vorgehensmodell bei allen Arten und Größen von Projekten skaliert. So schreibt Schwaber bspw., dass Scrum auch bei Projekten mit 800 Softwareentwicklern hervorragend skaliert[61]. Allerdings wird dabei verschwiegen, wie hoch der Aufwand ist, 100 Teams zu je acht Mitarbeitern zu koordinieren. Hinzu kommt, dass diese Teams mit dem Auftraggeber zu koordinieren sind, denn von ihm stammen die Anforderungen, die die Teams umzusetzen haben.

Das vorgestellte hybride Modell erhebt diesen Anspruch nicht. Es lässt sich auf Festpreisprojekte und anschließende Wartungsprojekte anwenden. Die Autoren gehen derzeit davon aus, dass das Modell bis zu einer mittleren Projektgröße (ca. 5 PM Aufwand, vgl. Kap. 2.3) skaliert. Für größere Projekte ist eine zusätzliche Koordinationsstelle für die Integration der Software erforderlich. Je mehr Softwareentwickler auf einer Codebasis arbeiten und Code in das SCM einspielen, desto mehr werden Konflikte auftreten.

Eine vergleichbare Situation stellt sich bei der Integration einzelner Module auf den Testsystemen dar. Nur aufeinander abgestimmte Module absolvieren auf dem Testsystem erfolgreich alle Tests. Die Nutzung von Continuous Integration bei großen Projekten benötigt folglich noch einen unterstützenden Prozess, der diese Integration koordiniert und aufeinander abgestimmte prototypische Komponenten auf die Testsysteme ausliefert. Dieser unterstützende Prozess wird häufig als **Release-Management** oder **Integrations-Management** bezeichnet.

[61] Vgl. Schwaber (2004).

5.6 Unternehmenskultur

Linssen[62] hat gezeigt, dass agile Vorgehen eine Änderung der Unternehmenskultur voraussetzen (vgl. Kap. 3, Abschn. 3.5.5). Da das hybride Modell auf Praktiken aus der agilen Softwareentwicklung setzt, ist auch hier eine moderate Anpassung der Unternehmenskultur notwendig. Die folgenden Aspekte sollten in der Unternehmenskultur etabliert sein, um bei Festpreisprojekten nach dem vorgestellten hybriden Modell ein angemessenes Qualitäts- und Risikomanagement zu ermöglichen:

- **Kommunikation:** Klare und offene Kommunikation sorgt für Effizienz und unterstützt die Fokussierung auf das Ziel.
- **Vertrauen:** Damit der bürokratische Aufwand minimiert wird, ist Vertrauen notwendig. Dazu gehört Vertrauen in die eigene Leistungsfähigkeit, der Organisation und in das etablierte Vorgehen. Vertrauen zwischen allen Beteiligten in einem Projekt oder einer Organisation wächst, wenn Vereinbarungen und Versprechen termingerecht eingehalten werden[63].
- **Fehlerkultur:** Das frühzeitige Aufdecken von Fehlern steigert die Projektrendite und vermeidet teure Änderungen. Dazu empfiehlt sich der Einsatz eines Defect Reporting Tools. Fehler dürfen nicht zu unverhältnismäßigen Sanktionen oder Schuldzuweisungen führen, weil das Verschleierung und Absicherungsverhalten nach sich ziehen würde[64].

Wenn diese Aspekte in der Unternehmenskultur etabliert sind, lassen sich agile Praktiken, wie z. B. Daily Stand-Up Meeting und Review, erfolgreich umsetzen. Für das Risikomanagement sind sie von großer Bedeutung, um interne Risiken im Entwicklungsprozess und externe risikobehaftete Einflüsse auf Seiten des Produkts und des Auftraggebers zu identifizieren.

Vertrauen ist eine grundlegende Voraussetzung für die Qualitätssicherung und den KVP. Verbesserungsvorschläge werden nur dann von den Beteiligten vorgebracht, wenn das Vertrauen vorherrscht, dass die Vorschläge von den Verantwortlichen ernsthaft geprüft werden. Das Vertrauen in die Leistungsfähigkeit der Beteiligten hat einen Einfluss auf die Qualitätssicherung. Nur wenn den Projektverantwortlichen bewusst ist, was Mitarbeiter zu leisten im Stande sind, kann eine korrekte Aufwands- und Terminabschätzung gewährleistet werden. Gerät das Projekt unter Termindruck, leidet zuerst immer die Qualität der Software. Dies lässt sich anhand von Kennzahlen messen, die in Kap. 4.7 ff. vorgestellt worden sind.

---

[62] Vgl. Linssen (2010).

[63] Vgl. Henkel et al. (2011), S. 69.

[64] Vgl. ebd., S. 69.

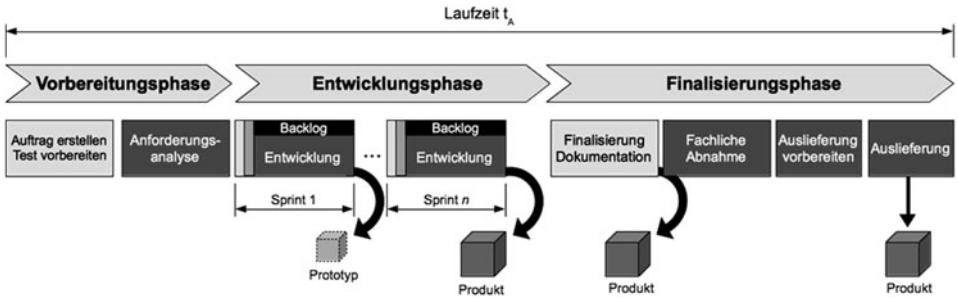

Abb. 5.7 Angepasstes Hybridmodell für Kleinstvorhaben. (Eigene Darstellung)

5.7 Besonderheiten bei der Bearbeitung von Kleinstvorhaben

Das hybride Modell soll neben mittleren Vorhaben auch auf sehr kleine Vorhaben anwendbar sein. Z. B. in der Software-Wartung, sind die Aufwendungen für Fehlerbehebungen und Änderungen teilweise sehr klein. Für sog. Kleinstvorhaben werden in diesem Kapitel die Besonderheiten genannt.

5.7.1 Veränderungen am hybriden Phasenmodell

Prinzipiell ist das in Kap. 5.1 vorgestellte hybride Phasenmodell auch zur Bearbeitung von Kleinstvorhaben anwendbar. Sie durchlaufen alle drei bekannten Phasen (Vorbereitungs-, Entwicklungs- und Finalisierungsphase). Jedoch können auf Grund des geringen Aufgaben-Umfangs einige Arbeitsschritte entfallen (Abb. 5.7).

Auch bei der Bearbeitung von Kleinstvorhaben dient die **Vorbereitungsphase** zur Erstellung eines Angebots. Die Teilphasen der Vorbereitungsphase (Anforderungsanfrage, Anforderungsanalyse, Angebotsabgabe) werden alle durchlaufen.

Die Anforderungsanfrage und die Anforderungsanalyse finden analog zu dem Vorgehen für mittlere Vorhaben statt. In Wartungsprozessen werden jedoch häufig Kontingente vorgehalten, um nicht jede einzelne Beauftragung zu klären. In diesem Fall entfällt unter Umständen eine Angebotsabgabe. Eine Anforderungsanalyse und eine Aufwandschätzung werden in diesem Fall aber dennoch durchgeführt.

Auch die Umsetzung in der Entwicklungsphase läuft wie im Vorgehen für mittlere Vorhaben ab. Nach dem groben **Systemdesign**, findet die eigentliche Umsetzung in Iterationen statt.

Bei einigen Vorhaben kann der Umsetzungsaufwand jedoch so gering sein, dass eine Umsetzung in Iterationen nicht sinnvoll ist. Aus diesem Grund wird zunächst die Größe des geplanten Vorhabens betrachtet. Insgesamt unterscheidet diese Betrachtung drei Szenarien:

1. **Das Kleinstvorhaben bildet ein eigenständiges Release:** Wartungsaufgaben können sehr kleinteilig sein. Der Aufwand zur Umsetzung beträgt z. B. einen PT oder noch weniger. Eine iterative Bearbeitung eines so kleinen Vorhabens wäre wenig praktikabel. Bei einem Kleinstvorhaben findet in der Entwicklungsphase nur eine einzige, sehr kurze Iteration statt.
2. **Mehrere Kleinstvorhaben bilden ein Release:** Setzt sich ein Release aus mehreren Kleinstvorhaben zusammen, dann wird jedes Vorhaben vollständig umgesetzt und zum Test bereitgestellt und erst nach dem Test das nächste Vorhaben begonnen. So werden die verschiedenen Kleinstvorhaben eines Releases iterativ abgearbeitet.
3. **Ein größeres Vorhaben bildet ein Release:** Wenn ein Vorhaben einen hohen Änderungsaufwand erfordert, so wird dieser in mehreren Iterationen bearbeitet. In der Literatur wird bei Scrum als typische Länge für Iterationen zwei bis vier Wochen genannt. Mit dem hybriden Modell für Kleinstvorhaben können aber auch Vorhaben mit einem Umsetzungsaufwand von unter 30 PT umgesetzt werden.
   Bei Vorhaben dieser Größe empfiehlt sich die Verkürzung der Iterationslänge. Dazu zerlegt man das Vorhaben in kleinere Arbeitspakete, die im Idealfall jeweils an einem einzelnen Arbeitstag umgesetzt werden. Nach jedem umgesetzten Arbeitspaket wird die Software zum Test bereitgestellt. Damit ein zeitnaher, fachlicher Test möglich ist, sollten die definierten Arbeitspakete fachliche Funktionen abbilden.

In der **Finalisierungsphase** gibt es bei der Bearbeitung von Kleinstvorhaben keine Besonderheiten zu berücksichtigen. Sie besteht aus den drei Arbeitsschritten **Finalisierung der Dokumentation**, **Kundenabnahme** und **Auslieferung**.

5.7.2 Methoden

Die in Kap. 5.2 vorgestellten Methoden können auch bei der Bearbeitung von Kleinstvorhaben verwendet werden.

Jedoch gilt es die folgenden Besonderheiten zu beachten:

- **Strukturierung:** Auch die Bearbeitung von Kleinstvorhaben profitiert von einem strukturierten Vorgehen. Im Einzelfall ist zu entscheiden, ob eine Zerlegung in kleinere Aufgaben sinnvoll ist.
- **Requirements Engineering:** Im hybriden Modell für Kleinstvorhaben ist die Erhebung von Anforderungen identisch mit dem hybriden Modell für mittlere Vorhaben. Allerdings lässt sich der Aufwand kleinere Aufgaben besser schätzen als der für große. Daher kann davon ausgegangen werden, dass die Schätzungen für Kleinstvorhaben zutreffender sind.
- **Kosten- und Aufwandsschätzung:** Hinsichtlich der Kosten- und Aufwandschätzung gibt es wenig Unterschiede im Vergleich zum hybriden Modell für mittlere Vorhaben. Auch für Kleinstvorhaben erstellt der Kunde ein Lastenheft, das alle funktionalen

und nicht-funktionalen Systemmerkmale enthält. Das detaillierte Lastenheft (inkl. Anwendungsfällen, Geschäftsobjekten, Prozessen, Benutzeroberflächen und System-Schnittstellen) dient anschließend als Grundlage für die Aufwandsschätzung. Die Unterschiede zwischen agilen und klassischen Methoden hinsichtlich der verwendeten Aufwandsschätzverfahren wurden bereits in Kap. 5, Abschn. 5.2.2 erläutert. Ebenso kann die Frage nach den richtigen Aufwandsschätzverfahren auch für Kleinstvorhaben nicht eindeutig beantwortet werden. Hier hängt die Auswahl der Methode von der Art und der Qualität der vom Auftraggeber gelieferten Anforderungen ab. Ein Vorteil bei der Bearbeitung von wiederkehrenden Themen (Pflege und Wartung existierender Software) ist, dass die Schätzung für eine neue Aufgabe leichter fällt, wenn eine vergleichbare Aufgabe schon einmal durchgeführt wurde.

- **Nutzenanalyse:** Auch Kleinstvorhaben sollten einen Nutzen erbringen. Die zugrunde liegenden Prinzipien (wie z. B. die Schwierigkeit den Nutzen monetär zu beziffern) unterscheiden sich hierbei nicht von denen von mittleren Vorhaben (siehe Kap. 5, Abschn. 5.2.3). Kleinstvorhaben werden häufig für Software-Wartung genutzt, daher gibt es eine Besonderheit zu berücksichtigen: Bei vielen Wartungstätigkeiten ist kein wirksamer Nutzen zu beobachten. Sie ähneln in dieser Hinsicht Infrastrukturmaßnahmen, wie z. B. Integrationen oder Migrationen. Bei sehr kleinen Vorhaben, die einen geschätzten Aufwand von wenigen Tagen haben, besteht darüber hinaus die Gefahr, dass die Nutzenanalyse mehr Aufwand als die Umsetzung selbst verursacht.
- **Prototyping und Feedbackmechanismen:** Der Vorschlag, dem Kunden bereits früh einen Prototypen zur Verfügung zu stellen und diesen kontinuierlich zu aktualisieren, gilt für sämtliche Vorhaben, unabhängig von ihrer Größe.
- **Test-Driven Development:** TDD kann auch bei Kleinstvorhaben genutzt werden, wenn für die Software-Wartung bereits eine hohe Abdeckung an Unit-Tests vorliegt. So kann sofort erkannt werden, wenn nach einer Anpassung, Teile der Software nicht mehr funktionieren.
- **Continuous Integration:** Prototyping und ATDD setzen voraus, dass dem Kunden stets eine aktuelle Version zum Test bereitgestellt wird. Continous Integration bietet eine gute Möglichkeit, dies zu erreichen.

Bei der Bearbeitung von Kleinstvorhaben laufen Qualitätsmanagement, Projektmanagement und Risikocontrolling ebenfalls parallel zum Prozess. Der Unterschied zu größeren Vorhaben liegt darin, dass es nicht immer wirtschaftlich ist, für jedes Kleinstvorhaben ein eigenes Risikocontrolling durchzuführen. In diesem Fall sollte diese Aufgabe zentral für alle Linien-Tätigkeiten vorgenommen werden.

5.7.3 Techniken und Werkzeuge

Neben den oben genannten Methoden existieren Techniken und Werkzeugen, die das hybride Modell für Kleinstvorhaben unterstützen. Sie ähneln denen für mittlere Vorhaben:

- **Product Backlog:** Ein Product Backlog kann wie beim hybriden Modell für mittlere Vorhaben genutzt werden. Für jede Software wird ein Product Backlog erstellt. Die Umsetzung von einzelnen Anforderungen aus dem Product Backlog stellt ein eigenes Kleinstvorhaben dar.
- **Checklisten:** Die Dokumentation von häufig durchgeführten Tätigkeiten in Checklisten, ist für unregelmäßig durchgeführte Kleinstvorhaben noch wichtiger, als im Rahmen eines größeren Vorhabens. Dadurch, dass zwischen der Bearbeitung von zwei Kleinstvorhaben längere Zeit verstreichen kann, besteht die Gefahr, dass wesentliche Details in Vergessenheit geraten.
- **Projektmanagement-Tool:** Prinzipiell kann ein Projektmanagement-Tool auch Vorteile bei der Bearbeitung von Kleinstvorhaben bieten. Jedoch kann der Aufwand für die Umsetzung des Kleinstvorhabens so gering sein, dass es sich nicht lohnt, einen entsprechenden Projektplan zu erstellen und zu pflegen.
- **Projektdatenbank:** Im Rahmen eines Kleinstvorhabens wird voraussichtlich keine eigene Projektdatenbank erstellt. Falls jedoch im ursprünglichen Projekt, das die Software erstellt hat, eine entsprechende Datenbank erstellt wurde, so können die in ihr enthaltenen Daten in der Wartung sinnvoll sein.
- **Defect Reporting Tool:** Auch im Rahmen von Wartungsprozessen kommen Defect Reporting Tools zum Einsatz. Tools wie Jira, Bugzilla oder FogBugz bilden sowohl Defects als auch Anforderungen ab.
- **Wiki:** Genau wie bei einer Projektdatenbank wird auch ein Wiki kaum für ein Kleinstvorhaben erstellt, jedoch kann ein im Projekt erstelltes Wiki in der Software-Wartung kann einWiki sinnvoll sein weiter genutzt werden.
- **Testsysteme:** Ohne geeignete Testsysteme ist vor dem produktiven Einsatz nicht sichergestellt, dass die Software einwandfrei funktioniert und die fachlichen Vorgaben erfüllt sind. Sie sind daher in jedem Software-Entwicklungsprozess unerlässlich.
- **Build Server:** Da CI und CD essentielle Methoden für alle Formen eines hybriden Vorgehens sind, ist ein Build Server unerlässlich.
- **Statische Analyse:** Statische Analysen, die Aussagen über die Code-Qualität erlauben, werden über den ganzen Software-Lebenszyklus durchgeführt. Somit stellen sie eine Technik dar, die in der Software-Wartung mit Hilfe des hybriden Modells durchgeführt wird.

5.7.4 Unterstützende Prozesse

Für das einzelne Kleinstvorhaben ist die Einrichtung unterstützender Prozesse möglicherweise unwirtschaftlich. Werden diese unterstützenden Prozesse jedoch zentral definiert, können sie anschließend für alle Kleinstvorhaben genutzt werden.

5.8 Zusammenfassung

Die Struktur des hybriden Modells eignet sich sehr gut zur Bearbeitung von Festpreisprojekten. Da zu Beginn des Projekts in Zusammenarbeit mit dem Kunden alle Anforderungen erhoben werden, kann ein Festpreis für die Umsetzung der Anforderungen vereinbart werden. Durch die Iteration in der Entwicklungsphase, die jeweils mit einem Test und einem Feedback des Kunden enden, werden Fehlentwicklungen vermieden.

Gängige Methoden aus der agilen Softwareentwicklung wie z. B. Continuous Integration (siehe Kap. 5, Abschn. 5.2.7) oder ATDD (siehe Kap. 5, Abschn. 5.2.6) unterstützen dieses Vorgehen.

Darüber hinaus ist das hybride Modell so flexibel, dass es auch für die Bearbeitung von Kleinstvorhaben, die z. B. regelmäßig bei der Software-Wartung auftreten, genutzt werden kann.

In den nachfolgenden beiden Kapiteln wird die Praxistauglichkeit des hybriden Modells durch zwei Fallstudien belegt.

Fallstudie numetris AG

6

Die nachfolgende Fallstudie betrachtet die Einführung des hybriden Modells in der numetris AG. Hierzu wird das Unternehmen und die Relevanz des Modells für die numetris AG vorgestellt. In Kap. 6.3 wird eine Ist-Analyse durchgeführt, um die Auswirkungen des hybriden Modells auf das Unternehmen abzuschätzen. Zur Beurteilung der Veränderungen aus Managementsicht dienen einerseits die vorangegangenen theoretischen Ausführungen und andererseits die Perspektiven einer Balanced Scorecard (BSC). Die BSC gibt dabei für die Fallstudie nur den organisatorischen Rahmen für die zu betrachtenden Unternehmensbereiche vor, die wie folgt definiert sind:

- **Interne Prozesse:** Betrachtet werden die unmittelbar mit der Erstellung der Software in Verbindung stehenden Geschäftsprozesse.
- **Lernen und Wachstum:** Hier werden die notwendigen Qualifikationen der Mitarbeiter den fachlichen Kompetenzen gegenübergestellt. Es ist zu überprüfen, inwiefern das hybride Modell mit seinen Methoden von den Mitarbeitern akzeptiert und angewendet wird.
- **Kunden:** Wie beeinflusst die Einführung des Modells die Interaktion mit Kunden?
- **Finanzwirtschaft:** Wie beeinflusst das neue Modell die Finanzwirtschaft des Unternehmens? Damit wird die Hypothese der Agilisten geprüft, dass inkrementelle Softwareentwicklung den Wert früher schafft.

Die Untersuchung in der Fallstudie prüft die in den Kap. 4 f. und 5 f. getroffenen Aussagen zur Flexibilität, zur Wertschöpfung, den eingesetzten Methoden und die Einhaltung der IT-Compliance im hybriden Modell. In der Zusammenfassung wird das Fazit unter Berücksichtigung der BSC-Perspektiven gezogen.

Gregor Sandhaus et al., *Hybride Softwareentwicklung,* Xpert.press,
DOI 10.1007/978-3-642-55064-5_6

6.1 Die numetris AG

Die numetris AG wurde 1996 gegründet und beschäftigt sich in der Energiewirtschaft mit Zählerfernauslesung und der Aufbereitung von Rohdaten bzw. Messwerten für die Netzsteuerung und die Abrechnung. Zu diesem Zweck wurde die Software-Suite enldamo geschaffen.

6.2 Relevanz des hybriden Modells für die numetris AG

Die numetris AG entwickelt kontinuierlich ihr Softwareprodukt enldamo weiter. Zurzeit erscheinen von dieser Software zwei Major-Releases jährlich, die aufgrund gesetzlicher Rahmenbedingungen beim Kunden zum 1. April und 1. Oktober installiert werden.

Die Weiterentwicklung der Software finanziert sich durch Kunden im Rahmen von Wartungsverträgen und Service Level Agreements (SLAs). Die Wartungspauschale wird in monatlichen Raten entrichtet.

Die Weiterentwicklung ist in Phasen von ca. fünf Monaten Entwicklung und jeweils einem Monat für das Release Management und Deployment getaktet. In diesen fünf Monaten Entwicklungsarbeit werden die vom Kundenstamm festgelegten Features umgesetzt, Bugs entfernt sowie durch den Gesetzgeber vorgegebene Prozesse und Regelungen in der Software verankert.

Neben dem Kernprodukt entwickelt die numetris AG individuelle Lösungen, die an das Kernprodukt angeschlossen werden oder eigenständig sind. Die Abrechnung individueller Lösungen erfolgt als Projekt zum Festpreis. Die Zahlung des vereinbarten Festpreises ist nach Abnahme fällig. Rückkopplung und Feedback finden kurz vor Auslieferung der Software statt. In seltenen Fällen finden Teilzahlungen der kompletten Projektsumme statt, sodass die numetris AG die meisten Projekte direkt oder indirekt vorfinanziert.

Die Einführung eines hybriden Modells hat für das Unternehmen den Vorteil, dass dasselbe Modell sowohl für die Weiterentwicklung des Kernprodukts als auch für Projekte auf Festpreisbasis angewendet wird. Das Modell und die zugrunde liegenden Prozesse sind nur einmalig dokumentiert, um die Compliance-Vorgaben der Kunden zu erfüllen.

Des Weiteren kann durch kleinere Phasen in der Softwareentwicklung schneller Feedback vom Kunden eingeholt werden. Dadurch werden fachliche Anforderungen sorgsamer und konkreter umgesetzt und technische Schulden[1] vermieden.

[1] Zum Thema Technische Schulden siehe Kap. 4, Abschn. 4.7.2 und die Erläuterungen zu den Technical Debt Points.

6.3 IST-Analyse

Zunächst wird die Ist-Situation vor Einführung des hybriden Modells erhoben. Dabei wird analysiert, ob das Modell ausreichend flexibel ist, um auf unterschiedliche Entwicklungsvorhaben angewendet zu werden. Weiterhin wird untersucht, ob diese Flexibilisierung grundsätzliche IT-Compliance Vorgaben verletzt. Abschließend wird die Wertschöpfung beurteilt.

6.3.1 Genutzte Methoden

Die numetris AG setzt bereits ein *Defect Reporting Tool* und einen *Daily Build*-Prozess für die automatische Erstellung der Software ein. Am Ende des Prozesses wird ein Installationspaket erstellt, das theoretisch direkt zum Kunden geliefert werden könnte.

Andere Methoden, wie sie im Lean software Development, Scrum oder XP genutzt werden, sind nur rudimentär vorhanden. Dazu gehört die Definition von Testfällen, die sich allerdings auf die Integration der Software beziehen. Das automatische Testen von Komponenten nach jeder Änderung (TDD) – der automatische Regressionstest nach einer Änderung – ist nicht umgesetzt.

6.3.2 Ermittlung der Flexibilität

Flexibilität kann aus zwei Perspektiven betrachtet werden. Die Eine stellt das Vorgehen bzw. den Prozess in den Vordergrund. Je kurzfristiger die Reaktion auf eine veränderte Anforderung ist, desto flexibler ist das Vorgehen.

Die andere Perspektive betrachtet die Fähigkeit des Modells unter verschiedenen Größen von IT-Vorhaben flexibel zu skalieren. Die Einführung des hybriden Modells soll den Aspekt des Tailoring weitgehend eliminieren.

Sowohl das bisherige Vorgehen für die Weiterentwicklung der Software als auch das Vorgehen in Projekten sind stark sequentiell organisiert. Kurzfristig geänderte Anforderungen sind aufgrund der vorlaufenden Planungsprozesse kaum umzusetzen.

6.3.3 Ermittlung der Wertschöpfung

Die Kosten für die Weiterentwicklung der Software en|damo sind durch die Umsätze mit den Wartungsverträgen gedeckt, sodass hier kein Handlungsbedarf besteht.

Bei Festpreisprojekten findet die Bezahlung häufig erst bei Abnahme durch den Kunden statt oder durch Vereinbarung von Teilzahlungen, wenn bestimmte Leistungen erfüllt sind. Diese Leistungskriterien orientieren sich an sequenziellen Modellen und sind daher definiert als:

- **Lieferung der Spezifikation:** Eine Teilzahlung erfolgt bei Lieferung der Spezifikation. Die Spezifikation dient später für die Abnahme der Software.
- **Lieferung der Software:** Die Software wird vollumfänglich geliefert und installiert.
- **Abnahme der Software:** Die Software wird durch den Kunden getestet und bei funktionaler Korrektheit abgenommen.

Die Auswirkungen des bisherigen Vorgehens sollen in der Ist-Analyse anhand von zwei Projekten dargestellt werden. Die Daten sind anonymisiert, die Rahmenbedingungen entsprechen der Realität. Kosten und Erlöse werden als Geldeinheiten (GE) angegeben. Das Verhältnis zwischen GE für Erlöse und Kosten entspricht ebenfalls der Realität, entspricht aber nicht den real erhobenen Stundensätzen. Bei der Kostenbetrachtung werden lediglich die Kosten für die im Projekt tätigen Mitarbeiter erhoben. Gemein- und Fixkosten sind nicht berücksichtigt.

Für beide Projekte gelten die folgenden Rahmenbedingungen:

- Pro Personentag (PT) werden 800 GE als Erlös veranschlagt.
- Ein PT entspricht dabei 6 Arbeitsstunden (AH) (0,75 Arbeitstage (AT)) produktive Zeit. Da ein Mitarbeiter in einem kreativen Beruf niemals zu 100 % produktiv ist.
- Die internen Kosten pro PT liegen bei 480 GE pro Mitarbeiter.
- Alle Projekte sind vorfinanziert, da Mittelabfluss (Kosten) und Mittelzufluss (Erlöse) nicht zeitgleich stattfinden. Sowohl die Mittelbeschaffung am Geldmarkt als auch die interne Vorfinanzierung wird verzinst und im Projektangebot berücksichtigt. Für die Darstellung wird ein marktüblicher Zinssatz von 7 % (Betriebsmittelkredit bzw. Geschäftskredit) veranschlagt.

Projekt 1 Hierbei handelt es sich um ein mittelgroßes Projekt mit 162 PT. Die Dauer des Projekts beträgt sieben Monate oder umgerechnet ca. 140 AT. Zwei Mitarbeiter arbeiten an dem Projekt, um den Endtermin einzuhalten. Drei Teilzahlungen sind mit dem Kunden vereinbart:

- 15 % bei Lieferung der Spezifikation
- 45 % bei Lieferung der Software
- 40 % bei Abnahme der Software

Projekt 2 Hierbei handelt es sich um ein kleines Projekt mit 10 PT Aufwand. Die Dauer des Projekts beträgt zwei Monate oder umgerechnet 40 AT. Das Projekt wird von einem Mitarbeiter durchgeführt. Eine Vorfinanzierung ist nicht notwendig.

Die Tab. 6.1 listet die Projektkalkulation der beiden zu vergleichenden Projekte auf. Die Wertstellung des Produkts aus bilanzieller Sicht erfolgt beim Kunden erst nach Abnahme der Software. Wird die Software als Festpreisprojekt unter der Prämisse eines Werkvertrags geliefert, so liegt das Risiko beim Ersteller. Folglich kann der Kunde das Produkt erst in

Tab. 6.1 Projektkalkulation für sequentielle Projekte

	Projekt 1	Erlösanteil	Projekt 2	Erlösanteil
Erlöse	129.600 GE		8.000 GE	
Zinsbelastung	3175 GE	2 %	196 GE	2 %
Kosten	77.760 GE		4.800 GE	
Gewinn	48.665 GE	38 %	3.004 GE	38 %

der Bilanz nach HGB § 248 II aktivieren, wenn das Produkt vollständig geliefert und abgenommen wurde, da sonst der Werkvertrag als nicht erfüllt gilt. Dauert ein Projekt mehrere Jahre bis zur Fertigstellung, erfolgt die Abschreibung erst nach Fertigstellung. Steuerliche Einspareffekte kommen somit erst spät zum Tragen.

6.3.4 Einhaltung der IT-Compliance

Für die Auslieferung und Installation der Software beim Kunden unterliegt die numetris AG den Vorgaben des ITIL™-Frameworks mit den umzusetzenden Prozessen *Service Support* und *Service Delivery*. Sowohl bei der Einführung eines neuen Releases als auch bei der Beseitigung von Fehlern dürfen das kundenseitige Change Management und Release Management nicht verletzt und vereinbarte SLAs müssen eingehalten werden.

Pro Jahr sollen nicht mehr als drei Major-Releases durchgeführt werden, da die Software vorab durch den Kunden aufwändig getestet wird. Diese Tests sind zusätzlich erforderlich, um den Anforderungen der Eichbehörden an die Software und die Dokumentation gerecht zu werden.

Mit der Fertigstellung des Produkts beginnt die Einleitung der Service Transition. Beim Kunden wird der nächste Rollout des Produkts geplant und terminiert. Dazu werden erforderliche Change Requests gestellt, von der IT verifiziert und vom Fachbereich später zur Durchführung freigegeben. Der Prozess sieht vor, dass das Produkt zunächst zur Qualitätssicherung in eine Integrationsumgebung ausgeliefert wird. In der Integrationsumgebung testet der Fachbereich neue Funktionalität und Bugfixes. Werden noch Fehler festgestellt, so werden diese von der numetris AG korrigiert und erneut über die IT des Kunden ein korrigiertes Produkt in die Integration ausgeliefert.

Ist die Validierung des Produkts ohne Beanstandung, erfolgt die Abnahme durch den Fachbereich und somit die Freigabe, dass das Produkt in den produktiven Betrieb übernommen wird. Der Prozess Service Transition wird also zweimal aus Sicht der Kunden-IT durchlaufen: Einmal für die Integrationsumgebung und ein zweites Mal für die Produktionsumgebung.

6.4 Veränderungen durch das hybride Modell

Um das hybride Modell vollumfänglich zu nutzen, sind organisatorische Veränderungen notwendig. Diese erfolgen in kleinen Schritten, um Mitarbeitern die Umstellung auf veränderte Methoden und neue Prozesse zu ermöglichen.

6.4.1 Optimierung der genutzten Methoden

Der Daily bzw. Nightly Build wurde durch Continuous Integration erweitert. Komponenten werden nach Modifikationen in das Konfigurationsmanagement zurückgespielt und durch das CI-System automatisch gebaut. Weiterhin werden statische Code-Analysen vorgenommen, um typische Programmierfehler aufzudecken. Sofern Unit-Tests für Komponenten definiert sind, werden diese ebenfalls ausgeführt. Legacy-Code unterliegt dabei nicht den Anforderungen an das TDD. Für alle neuen Anforderungen sollen nach Möglichkeit Unit-Tests geschrieben werden und automatische Integrationstests definiert sein.

Das Defect Reporting Tool wird zur Erfassung der Anforderungen und als Product Backlog genutzt. Über einen Mechanismus zur Darstellung eines Kanbans können die zurzeit in Arbeit befindlichen Funktionalitäten und Bugfixes je Mitarbeiter eingesehen werden. Das Defect Reporting Tool berechnet jede Nacht ein Burndown-Chart pro Mitarbeiter, sodass frühzeitig Überlastungen identifiziert und notwendige Maßnahmen ergriffen werden können.

Weitere Methoden, die mit der Einführung des hybriden Modells Einzug gehalten haben, sind:

- **Daily Stand-Up Meeting:** Im Gegensatz zur Definition bei Scrum ist die Gesamtzeit auf eine Stunde statt 15 min. limitiert, weil Entwickler mit Problemen bei der Fehleranalyse oder komplexen Lösungsansätzen ihre Ideen sofort diskutieren dürfen.
- **Real Customer Involvement:** Alle Kunden, die das Produkt im Einsatz haben, können neue Funktionen für einen Sprint auswählen. Die Kommunikation mit den Kunden findet direkt vor Ort oder in Telefonkonferenzen statt.
- **Ten-Minute Build**: Dauerte der Daily Build vormals fast eine Stunde, um alle Komponenten zu bauen und zu assemblieren, so werden durch das CI-System einzelne Komponenten gebaut, die sofort zum Test bereit stehen. Die durchschnittliche Zeit für den Bau von Komponenten und die anschließende Ausführung von Tests beträgt zurzeit ca. drei Minuten.
- **Incremental Design:** Neue Komponenten werden nach objektorientierten Entwurfsmustern entwickelt, die einen inkrementellen Aufbau zulassen. Für ältere Komponenten wird dies nur vorgenommen, wenn ein grundlegendes Refactoring erforderlich ist.

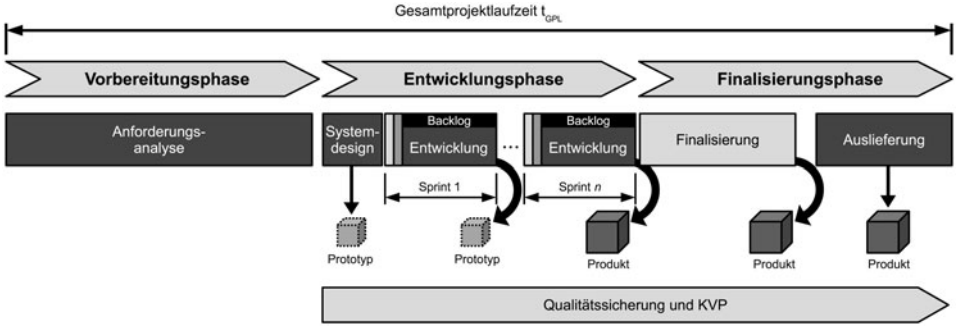

Abb. 6.1 Angepasstes Modell für die Wartung eines bestehenden Produkts. (Eigene Darstellung)

6.4.2 Erhöhung der Flexibilität

Die Entwicklung und Einführung des hybriden Modells ermöglicht es Mitarbeitern, auf Veränderungen zügiger zu reagieren. Dies wird weniger durch den Prozess getrieben als durch die eingeführten Methoden.

Darüber hinaus hat das hybride Modell einen weiteren Vorteil bzgl. Flexibilität, da es sich sowohl für Projekte als auch für interne Entwicklungsvorhaben mit geringem Tailoring-Aufwand einsetzen lässt. Lediglich die Vorbereitungsphase ist anzupassen (vgl. Abb. 6.1).

Betrachtet man das hybride Modell als Prozessmodell für verschiedene IT-Vorhaben und als Basis für eine Zertifizierung nach CMMI oder ISO 9001, so werden die Prozesse nur für ein Prozessmodell beschrieben, denn das Prozessmodell kann für unterschiedliche Arten von IT-Vorhaben (vgl. Kap. 2.5) genutzt werden. Diese Zertifizierungen sind z. B. erforderlich, um an internationalen Ausschreibungen teilnehmen zu können und die numetris AG erspart sich die Verdopplung der internen Prozessdokumentation.

Die Prozessbeschreibung des hybriden Modells hat keinen Vorteil, wenn ein potenzieller Auftraggeber ein Vorgehen im Projekt nach V-Modell XT verlangt. Ob das V-Modell XT nach bestimmten Tailoring-Maßnahmen möglicherweise identisch mit den Prozessschritten und Methoden des hybriden Modells wäre, wurde in der Fallstudie nicht weiter untersucht.

6.4.3 Ermittlung der Wertschöpfung

Zur Ermittlung, ob das hybride Modell die Zahlungsströme optimieren kann, wird ein vom Zeitraum her mit Projekt 1 vergleichbares Projekt H1 durchgeführt.

Projekt H1 Die Dauer des Projekts beträgt sieben Monate oder umgerechnet ca. 140 AT. Zwei Mitarbeiter sind mit dem Projekt beschäftigt. Der Aufwand beträgt 45 PT. Es gelten die Annahmen über die Kostenstrukturen aus der Ist-Analyse. Mit dem Kunden sind folgende Teilzahlungen vereinbart:

Tab. 6.2 Projektkalkulation für hybride Projekte

	Projekt H1	Erlösanteil
Erlöse	36.000 GE	
Zinsbelastung	882 GE	2 %
Kosten	21.600 GE	
Gewinn	13.518 GE	38 %

- 40 % bei Lieferung des Prototypen
- 60 % bei Lieferung des fertigen Produkts

Das Projekt wird aufgrund der Dauer vorfinanziert.

Die Tab. 6.2 listet die Projektkalkulation anhand der oben genannten Parameter für das hybride Projekt H1 auf.

Aufgrund veränderter Teilzahlungen können die Zahlungsströme auf Seiten des Herstellers nicht optimiert werden. Anders sieht es bei der Wertstellung für die Bilanzierung aus. Sofern der Prototyp bereits funktional ist, kann der Kunde bereits die Kosten für die Erstellung des Prototyps aktivieren und somit abschreiben.

6.4.4 Einhaltung der IT-Compliance

Die IT-Compliance wird durch die sequentielle Gestaltung der Finalisierungsphase eingehalten und steht nicht im Konflikt zu Anforderungen aus dem ITIL™ -Framework an den Prozess Service-Transition. Die nachfolgende Abb. 6.2 verdeutlicht das Zusammenspiel.

Die Einführung des hybriden Modells hat auf die Einhaltung der IT-Compliance keinen Einfluss. Der Prozess wird wie im IST-Zustand beschrieben weiter durchgeführt.

Die Rückkopplung durch den Kunden während der Sprints in der Entwicklungsphase und entsprechende Fabriktests haben Beanstandungen und vollständige Fehlentwicklungen bei neuen Funktionen deutlich reduziert. Durch die frühzeitige Entdeckung von Diskrepanzen zwischen Anforderungen und implementierter Funktionalität werden die Abnahmephase und umfangreiche Tests am Ende verkürzt.

6.5 Zusammenfassung und Ausblick

Die Fallstudie über die Einführung des hybriden Modells bei der numetris AG zeigt, dass die Anpassung der internen Prozesse mit minimalem Aufwand möglich ist. Mit der Einführung des Modells wurden automatisch auch die aus Lean Software Development, Scrum oder XP bekannten Methoden übernommen.

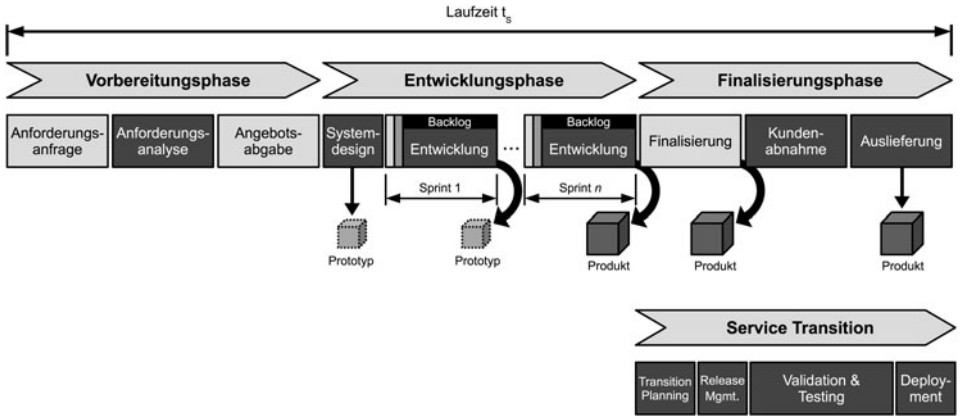

Abb. 6.2 Service-Transition im hybriden Modell. (Eigene Darstellung)

Zeitgleich intensivierte sich die Interaktion mit dem Kunden. Die Reaktion der Entwickler auf kurzfristige Änderungen bei den Anforderungen oder der Reihenfolge der Prioritäten ist flexibler, da bei der Weiterentwicklung der Software en|damo Prioritäten und Anforderungen monatsweise erfasst werden.

Aus finanzwirtschaftlicher Sicht führt das Modell nicht zwingend zu optimierten Zahlungsströmen für die numetris AG. Aber Kunden können Prototypen in der Bilanz frühzeitig aktivieren. Durch die Aufnahme des immateriellen Vermögensgegenstandes auf der Aktivseite, besteht die Möglichkeit, die Kosten bereits frühzeitig abzuschreiben.

Das Modell verstößt nicht gegen Anforderungen der IT-Compliance und hält vertragliche Regelungen ein.

Zukünftig plant die numetris AG, Zahlungsströme und Vorfinanzierung von Produkten zu optimieren.

Für den Kunden ist die Lieferung eines Prototypen ebenso eine Messgröße an der sich Zahlungen festmachen lassen, wie die Lieferung von Dokumenten. Für Fachanwender hat die Lieferung eines funktionierenden Prototypen einen höheren Wert, als eine gedruckte Spezifikation. Somit ist der Fachanwender auch eher bereit, Teilzahlungen zu leisten.

Durch monatliche Zahlungen, die an der Auslieferung von funktionalen Prototypen festgemacht werden, lässt sich die Vorfinanzierung deutlich reduzieren.

Fallstudie DEVK

7

Das folgende Fall-Beispiel zeigt, dass das hybride Modell gegenüber einer klassischen Vorgehensweise Vorteile hinsichtlich Flexibilität und Durchlaufzeiten hat. Diese Betrachtung erfolgt am Beispiel der DEVK Versicherungen.

Dazu wird zunächst das Unternehmen DEVK Versicherungen und der Software-Wartungsprozess *Auftragsbearbeitung* vorgestellt und hinsichtlich Flexibilität, Durchlaufzeit und die Einhaltung von IT-Compliance analysiert. Anschließend zeigt eine theoretische Betrachtung, wie sich der Prozess durch die Einführung des hybriden Modells verändern könnte und analysiert die Auswirkungen auf die oben genannten Faktoren.

7.1 Die DEVK Versicherungen

Am 1. April 1886 wurde die *Sterbekasse der Beamten und Arbeiter im Bezirke der Königlichen Eisenbahndirektion zu Breslau* gegründet, die als direkter Vorläufer des heutigen DEVK Lebensversicherungsvereins gilt.[1]

Die DEVK betreut bundesweit rund 4 Mio. Kunden mit 13,4 Mio. Risiken in allen Versicherungssparten. Zu ihr gehören ca. 1250 Geschäftsstellen, mit ca. 2250 hauptberuflichen Vertriebspartnern und ca. 3400 nebenberufliche Vermittler.[2]

Nach eigenen Angaben ist die DEVK auf Basis der Vertragsanzahl Deutschlands viertgrößter Hausrat- sowie fünftgrößter Pkw- und Haftpflichtversicherer.[3]

[1] Vgl. DEVK – Geschichte (2013).

[2] Vgl. DEVK (2013).

[3] Vgl. DEVK – Geschichte (2013).

Gregor Sandhaus et al., *Hybride Softwareentwicklung,* Xpert.press,
DOI 10.1007/978-3-642-55064-5_7

7.2 Relevanz der Thematik für die DEVK

Die DEVK hat sich gemäß ihrer IT-Strategie die schnelle Reaktion auf Kundenbedürfnisse in der Produktgestaltung, z. B. mit neuen Tarifmodellen, als Ziel gesetzt. Neben diesen Anpassungen gibt es für die DEVK als Versicherungsunternehmen auch immer wieder sogenannte „Muss"-Themen, wie z. B. SEPA (Single Euro Payments Area). Bei SEPA handelte es sich um ein Programm der Europäischen Union, das zum Ziel hat, europaweit einheitliche Verfahren und Standards für die Abwicklung von Euro-Zahlungen zu schaffen. Da die neuen Verfahren seit dem 01.02.2014 verbindlich sind, war die DEVK gesetzlich dazu verpflichtet, bis zu diesem Termin alle Systeme, die am Zahlungsverkehr teilnehmen, entsprechend anzupassen.

Eine flexible und schnelle Anpassung der bestehenden Software-Systeme steht daher immer mehr im Fokus der DEVK.

7.3 Der Software-Wartungsprozess Auftragsbearbeitung

Die DEVK hat einen eigenen Softwareentwicklungsprozess zur Bearbeitung von Kleinstvorhaben – die sogenannte *Auftragsbearbeitung* – etabliert. Gemäß der Prozessdokumentation dient das Vorgehensmodell Auftragsbearbeitung zur „Bearbeitung von kleinen und komplexen Aufträgen. Darüber hinaus wird die Optimierung und Korrektur von in Betrieb befindlichen Systemen durchgeführt (außer Notfallfehler)." Es handelt sich also um einen Wartungsprozess für Software. Abbildung 7.1 zeigt den schematischen Ablauf des Vorgehensmodells.

Die in der FB-Zeile dargestellten Arbeitspakete werden vom Fachbereich (z. B. einem Versicherungspartner einer der Versicherungssparten) durchgeführt. In der mit AO beginnenden Zeile handelt es sich um Arbeitspakete die von der Anwendungsoptimierung, eine der vier IT-Abteilungen, durchgeführt werden. IB/IE bezieht sich auf die Auslieferung durch die IT-Infrastruktur.

Wie aus der Abbildung zu erkennen sind im Prozess Rücksprünge möglich. Ändern sich beispielsweise während der Umsetzung die Anforderungen, dann wird der Auftrag zurück an den Fachbereich gegeben, die Anforderungen angepasst und der Prozess erneut durchlaufen.

Die einzelnen Aufträge unterscheiden sich in ihrem Umfang. So gibt es viele Aufträge mit einem IT-Aufwand von nur einem Tag. Es existieren aber auch Aufträge, deren Umsetzung mehrere Wochen in Anspruch nehmen kann.

In den folgenden Kapiteln wird der Prozess Auftragsbearbeitung in Hinblick auf die drei Aspekte Durchlaufzeit, Flexibilität und Einhaltung von IT-Compliance untersucht. Bei den Analysen hinsichtlich Durchlaufzeit und Flexibilität wird ein Auftrag mit einem geschätzten Umsetzungsaufwand von 1 PT betrachtet, da Aufträge mit diesem geschätzten Aufwand am häufigsten vorkommen (ca. 36,5 % aller Aufträge).[4]

[4] Hierbei gilt es zu beachten, dass diese 1 PT nur den Umsetzungsaufwand der IT darstellen. Hinzu kommen noch Aufwände für die Fachbereiche und die IT-Infrastruktur (siehe Abschn. 7.3.2).

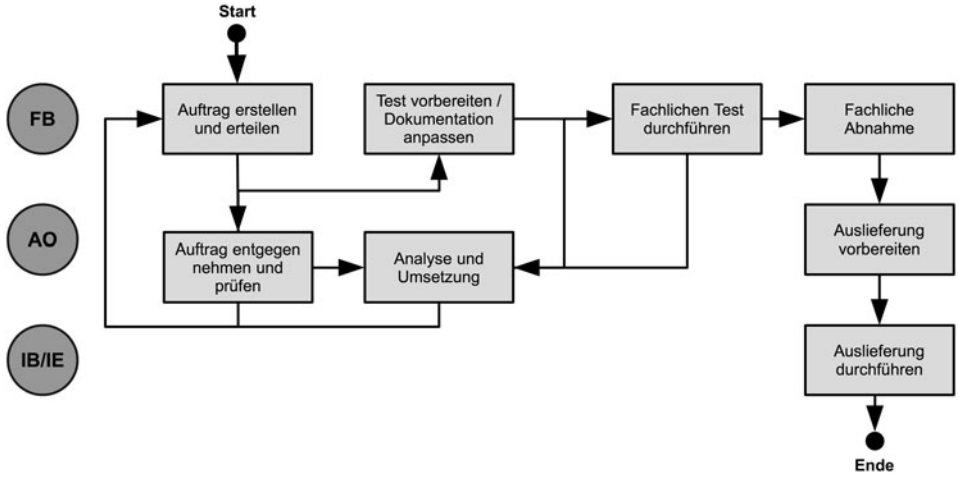

Abb. 7.1 Teilprozesse des Vorgehensmodells Auftragsbearbeitung. (Eigene Darstellung)

7.3.1 Ermittlung der Durchlaufzeit

Die Durchlaufzeit des Prozesses Auftragsbearbeitung einschließlich der enthaltenen Wartezeiten wird über das verwendete Workflow-Management System ermittelt. Bei der Bearbeitung eines Auftrags werden an definierten Punkten Zeiten gespeichert, mit deren Hilfe sich die durchschnittlichen Durchlaufzeiten der einzelnen Prozessphasen ermitteln lassen. Tabelle 7.1 stellt die Durchlaufzeiten der Arbeitspakete in Tagen und Stunden dar:

Die durchschnittliche Durchlaufzeit für einen Auftrag mit 1 PT geschätzten IT-Aufwand durch den gesamten Prozess beträgt demnach 33,38 Kalendertage. Auffallend ist der hohe Anteil der gemessenen Wartezeiten. Die Summe aller Wartezeiten zwischen den Arbeitsschritten beträgt 17,88 Kalendertage, was somit über 50 % der gesamten Durchlaufzeit ausmacht.

Da es sich bei den angegebenen Zeiten um Kalendertage handelt, fallen auch Wochenenden, an denen bei der DEVK in der Regel nicht gearbeitet wird, in die gemessenen Zeiträume. Die Wartezeit von zwei Tagen für ein Wochenende kann in jeder Phase auftreten. Daher wird im Folgenden davon ausgegangen, dass Wochenenden gleichmäßig in allen Phasen enthalten sind und werden nicht weiter berücksichtigt.

7.3.2 Ermittlung der Flexibilität

Wie in Kap. 2.7 definiert, können flexible Prozesse auf unterschiedliche oder geänderte Anforderungen zeitnah reagieren. Um die Flexibilität zu quantifizieren, wird untersucht, wie viel zusätzlicher Aufwand erforderlich wäre, wenn sich die Anforderungen während der Bearbeitung ändern. Hierzu wird in zwei Szenarien unterschieden:

Tab. 7.1 Durchschnittliche Durchlaufzeiten für einen 1 PT-Auftrag. (Eigene Darstellung)

Teilprozesse	Durchschn. Durchlaufzeit (t)	Durchschn. Durchlaufzeit (h)
Auftrag erstellen und erteilen	0,38	9,14
Wartezeit vor der Annahme	1,78	42,75
Auftrag entgegen nehmen u. prüfen	1,35	32,43
Wartezeit vor der Bearbeitung	8,10	194,52
Analyse/ Umsetzung	4,12	98,82
Wartezeit vor dem fachlichem Test	5,79	139,07
Fachlicher Test	1,73	41,49
Wartezeit vor der fachlichen Abnahme	2,21	53,08
Fachliche Abnahme	0,65	15,61
Auslieferung	7,26	174,16
Summe	*33,38*	*801,07*

1. Während der Bearbeitung des Auftrags ändern sich die Anforderungen nicht und beim fachlichen Test werden keine Fehler gefunden. Aus diesem Grund werden alle Teilprozesse nur einmal durchlaufen.
2. Während des fachlichen Tests stellt sich heraus, dass die aufgeschriebenen Anforderungen nicht den tatsächlichen Wünschen des Fachbereichs entsprechen. Der Auftrag wird zurückgegeben, die Anforderungen angepasst und der Prozess erneut durchlaufen. Dies führt dazu, dass Teilprozesse doppelt durchlaufen werden.

Zur Bestimmung der Prozesskostensätze beider Szenarien sind für jeden Arbeitsschritt die benötigten Arbeitsaufwände bekannt. Da diese zu Beginn der Analyse nicht vorlagen, wurden diese durch eine Umfrage ermittelt. Im Rahmen dieser Umfrage wurden ca. 450 Personen befragt. Aus den erhaltenen Antworten lassen sich die durchschnittlichen Aufwände für die einzelnen Teilprozesse ermitteln. Tabelle 7.2 stellt diese in einer Übersicht dar.

Der durchschnittliche Aufwand, der für die Bearbeitung eines Auftrags mit geschätztem 1 PT IT-Aufwand benötigt wird, beträgt demnach 22,82 h.

Mit den ermittelten durchschnittlichen Aufwänden werden nun die Prozesskosten für die beiden definierten Szenarien ermittelt. Tabelle 7.3 zeigt die Prozesskosten für einen Durchlauf der Auftragsbearbeitung, bei dem sich die Anforderungen nicht ändern (Szenario 1).

Wenn sich die Anforderungen während des Prozesses ändern (Szenario 2), dann steigen die Prozesskosten wie in Tab. 7.4 dargestellt.

Tab. 7.2 Durchschnittlicher Aufwand für die definierten Teilprozesse. (Eigene Darstellung)

Teilprozess	Durchschnittlicher Aufwand (h)
Auftrag erstellen und erteilen	1,05
Auftrag entgegen nehmen u. prüfen	0,51
Auftrag analysieren u. umsetzen	8,87
Technische Dokumentation anpassen	0,97
Fachliche Dokumentation anpassen	1,48
Fachlichen Test vorbereiten	1,69
Fachlichen Test durchführen	2,52
Fachliche Abnahme durchführen	0,5
Auslieferung vorbereiten	2,95
Staging inkl. Produktivsetzung	2,28
Gesamt	*22,82*

Tab. 7.3 Prozesskosten eines einfachen Durchlaufs. (Eigene Darstellung)

Teilprozess	Aufwand (h)	Kostensatz (€)	Kosten/ Durchführung (€)	Prozessmenge	Kosten (€)
Auftrag erstellen u. erteilen	1,05	63,00	66,15	1	66,15
Auftrag entgegennehmen und prüfen	0,51	78,00	39,78	1	39,78
Auftrag analysieren u. umsetzen	8,87	78,00	691,86	1	691,86
Technische Dokumentation anpassen	0,97	78,00	75,66	1	75,66
Fachliche Dokumentation anpassen	1,48	63,00	93,24	1	93,24
Fachlichen Test vorbereiten	1,69	63,00	106,47	1	106,47
Fachlichen Test durchführen	2,52	63,00	158,76	1	158,76
Fachliche Abnahme durchführen	0,5	63,00	31,50	1	31,50

Tab. 7.3 (Fortsetzung)

Teilprozess	Aufwand (h)	Kostensatz (€)	Kosten/ Durchführung (€)	Prozessmenge	Kosten (€)
Auslieferung vorbereiten/ Staging	2,95	78,00	230,10	1	230,10
Staging inkl. Produktivsetzung	2,28	78,00	177,84	1	177,84
Gesamt	*22,82*		*1.671,36*	*1*	*1.671,36*

Tab. 7.4 Prozesskosten eines Durchlaufs mit ändernden Anforderungen. (Eigene Darstellung)

Teilprozess	Aufwand (h)	Kostensatz (€)	Kosten/ Durchführung (€)	Prozessmenge	Kosten (€)
Auftrag erstellen u. erteilen	1,05	63,00	66,15	2	132,30
Auftrag entgegennehmen und prüfen	0,51	78,00	39,78	2	79,56
Auftrag analysieren u. umsetzen	8,87	78,00	691,86	2	1.383,72
Technische Dokumentation anpassen	0,97	78,00	75,66	2	151,32
Fachliche Dokumentation anpassen	1,48	63,00	93,24	2	186,48
Fachlichen Test vorbereiten	1,69	63,00	106,47	2	212,94
Fachlichen Test durchführen	2,52	63,00	158,76	2	317,52
Fachliche Abnahme durchführen	0,5	63,00	31,50	1	31,50
Auslieferung vorbereiten/ Staging	2,95	78,00	230,10	1	230,10
Staging inkl. Produktivsetzung	2,28	78,00	177,84	1	177,84
Gesamt	*22,82*		*1.671,36*	*1*	*2.903,28*

Der Prozesskostensatz für die Bearbeitung eines Auftrags mit geschätztem IT-Aufwand von 1 PT, bei dem sich die Anforderungen während der Bearbeitung nicht ändern, beträgt laut den Berechnungen in Tab 7.4 1.671,36 € und bei sich ändernden Anforderungen 2.903,28 €. Das bedeutet, dass die Prozesskosten bei sich ändernden Anforderungen um ca. 73 % über denen des optimalen Durchlaufs liegen.

7.3.3 Einhaltung von IT-Compliance

Für Versicherungsunternehmen gelten eine Reihe von IT-Compliance Anforderungen. Im Rahmen dieser Fallstudie werden die Grundsätze ordnungsmäßiger DV-gestützter Buchführungssysteme (GoBS) als Beispiel für eine allgemeingültige Vorschrift und die Mindestanforderungen an das Risikomanagement (MaRisk) als Beispiel für eine branchenspezifische Vorschrift betrachtet. Im Folgenden werden die Anforderungen aus diesen beiden Vorschriften, die Auswirkungen auf einen Softwareentwicklungsprozess haben, herausgestellt.

7.3.3.1 Grundsätze ordnungsmäßiger DV-gestützter Buchführungssysteme

„Die Grundsätze ordnungsmäßiger DV-gestützter Buchführungssysteme sind von der deutschen Finanzverwaltung aufgestellte Regeln zur Buchführung mittels Datenverarbeitungssystemen."[5]

Rechnungslegungsrelevante Systeme unterliegen nach dem HGB den Grundsätzen ordnungsmäßiger Buchführung (GoB).[6] Diese allgemeinen Anforderungen an die Buchführung wurden mit Schreiben des Bundesministeriums für Finanzen vom 07.11.1995 „Grundsätze ordnungsmäßiger DV-gestützter Buchführungssystemen (GoBS)" für eine DV-gestützte Buchführung präzisiert.[7]

Die GoBS richten sich im eigentlichen Sinne nur an Buchführungssysteme. Im Laufe der letzten Jahre hat sich jedoch die Betrachtungsweise auf DV-gestützte Buchführung geändert. Die „Buchhaltung" ist nicht mehr klar von anderen Bereichen abgrenzbar. Durch die zunehmende Integration von DV-Systemen können „Buchhaltungsdaten" in verschiedenen Arbeitsabläufen, die nicht der Abteilung „Buchhaltung" zugeordnet sind, entstehen. Dies führt dazu, dass diese Systeme eine Belegfunktion erlangen und sie damit ebenfalls den GoBS unterliegen.[8]

Die meisten Anforderungen der GoBS richten sich als Anforderung an die genutzten IT-Systeme, wie z. B. die Möglichkeit zur Erbringung eines sachlichen und zeitlichen Nachweis über sämtliche buchführungspflichtigen Geschäftsvorfälle.[9] Auch finden sich unter

[5] Laudon (2010), S. 878.

[6] Siehe §§ 238, 239, 257 und 261 HGB.

[7] Vgl. Gaulke (2010), S. 184.

[8] Vgl. GoBS (1995), S. 6.

[9] Vgl. ebd., S. 8.

ihnen Anforderungen an Datensicherheit[10], Aufbewahrungsfristen[11] und die Entwicklung und Anpassung von Software. Dazu zählen:

- **Maßnahmen zur Sicherung der Programmidentität:** Als Programmidentität definieren die GoBS, dass „das in der Dokumentation beschriebene Verfahren dem in der Praxis eingesetzten Programm (. . .) voll entspricht (. . .).“[12] Das bedeutet, dass ein Programm zu dokumentieren ist, und diese Dokumentation mit dem produktiv eingesetzten Stand übereinstimmt.
  Gemäß der GoBS ist eine der Voraussetzungen für die Sicherstellung der Programmidentität, dass es Richtlinien für die Programmierung, Programmtests, Programmfreigaben und Programmänderungen gibt.[13]
  Um den generellen Anforderungen an Transparenz, Kontrollierbarkeit und Verlässlichkeit zu entsprechen, ist zu gewährleisten, dass alle produktiv eingesetzten Programme autorisiert sind und für den richtigen Zweck eingesetzt werden.[14] Wie diese Richtlinien und Autorisierungsverfahren genau auszusehen haben, gibt die GoBS nicht vor.
- **Erstellung einer Verfahrensdokumentation:** Die DV-Buchführung soll einem sachverständigen Dritten ermöglichen, in angemessener Zeit die formelle und sachliche Richtigkeit zu prüfen. Darum ist es erforderlich, eine entsprechende Dokumentation zu erstellen. An die formale Gestaltung der Verfahrensdokumentation sind keine Anforderungen gestellt, aber sie hat z. B. die folgenden Bestandteile aufzuweisen:
  – Beschreibung der sachlogischen Lösung
  – Beschreibung der programmtechnischen Lösung
  – Beschreibung, wie die Programmidentität gewahrt wird[15]

  Die sachlogische Beschreibung schildert die fachlichen Aufgaben aus Sicht des Anwenders und die programmtechnische Beschreibung zeigt, wie diese in den Programmen umgesetzt wurden.[16] Das erfordert eine Dokumentation aus Sicht der Anwender und eine technische Dokumentation der Systeme. Außerdem sind alle Programmänderungen zu dokumentieren. Dies kann sowohl über eine organisatorische Maßnahme als auch automatisch erfolgen.
  Zu der geforderten Beschreibung, wie die Programmidentität gewahrt wird, gehört eine Beschreibung des Freigabeverfahrens inkl. der Freigabekompetenzen und der zu durchlaufenden Tests. Die Freigabeerklärung ist zu dokumentieren. Aus ihr ist ersichtlich, ab welchem Zeitpunkt welche Programmversion für den produktiven Einsatz vorgesehen ist.[17]

[10] Vgl. GoBS (1995), S. 12.
[11] Vgl. ebd., S. 16.
[12] Ebd., S. 8.
[13] Vgl. ebd., S. 11.
[14] Vgl. ebd., S. 12.
[15] Vgl. ebd., S. 14.
[16] Vgl. ebd., S. 14 f.
[17] Vgl. GoBS (1995), S. 16.

7.3.3.2 Mindestanforderungen an das Risikomanagement

„Die Bundesanstalt für Finanzdienstleistungsaufsicht (BaFin) macht in den Mindestanforderungen an das Risikomanagement (MaRisk) Vorgaben u. a. für die Sicherheit der IT-Systeme, das Outsourcing sowie die IT-Notfallplanung bei Banken (MaRisk 2010), Versicherungen (MaRisk VA 2009) und Investmentgesellschaften (InvMaRisk 2010).“[18] Auch wenn diese drei sich mit vergleichbaren Themenstellungen beschäftigen, unterscheiden sie sich in Struktur und Inhalt. Selbst in gleichen Themenfeldern unterscheiden sie sich oftmals in Formulierungen.[19] Da der Fokus dieser Fallstudie auf der DEVK als Versicherungsunternehmen liegt, wird nur die MaRisk VA genauer analysiert. Wenn im Folgenden der Begriff „MaRisk“ verwendet wird, ist damit die „MaRisk VA“ gemeint.

Die aktuellste Version der MaRisk wurde von der BaFin im Rundschreiben vom 22.01.2009 veröffentlicht. Nach eigenen Angaben gibt das Rundschreiben „einen flexiblen und praxisnahen Rahmen für die Ausgestaltung des Risikomanagements der beaufsichtigten Unternehmen, Gruppen und Finanzkonglomerate vor.“[20] Da es sich um Mindestanforderungen handelt, steht es Versicherungsunternehmen frei, sich selbst höhere Standards aufzuerlegen.[21] Die enthaltenen Anforderungen gelten u. a. für Versicherungsunternehmen im Sinne des § 105 VAG.[22]

Im Rahmen dieses Buches sind lediglich die Anforderungen, die an die IT-Systeme eines Versicherungsunternehmens gestellt werden, relevant. Diese befinden sich im Kap. 7, Abschn. 7.2.2.2 „Betriebliche Anreizsysteme und Ressourcen“ des oben genannten Rundschreibens. Gefordert werden die folgenden Aspekte:

- Um die Integrität und die Verfügbarkeit von IT-Systemen und IT-Prozessen sicherzustellen, hat sich deren Ausgestaltung an gängigen Standards (z. B. BSI Grundschutz) zu orientieren.
- Vor ihrem erstmaligen Einsatz und nach wesentlichen Veränderungen sollten IT-Systeme getestet und von den fachlich und technisch zuständigen Mitarbeitern abgenommen werden.
- Eine grundsätzliche Trennung von Produktions- und Testumgebungen ist sicherzustellen.
- Die fachlich und technisch verantwortlichen Mitarbeiter sind an der Entwicklung und Änderung von programmtechnischen Vorgaben zu beteiligen. Der Anwender darf die Software selbst nicht freigeben.

---

[18] Müller (2011), S. 13.

[19] Vgl. ebd., S. 36.

[20] MaRisk VA (2009), S. 3.

[21] Vgl. ebd., S. 4.

[22] Vgl. ebd., S. 5.

- Jede vom Unternehmen eingesetzte Software hat den Anforderungen des MaRisk-Rundschreibens zu genügen. Dies bezieht sich sowohl auf selbst erstellte, als auch auf gekaufte Software.[23]

7.3.3.3 Einhaltung der ermittelten Anforderungen

In Tab. 7.5 werden die erarbeiteten IT-Compliance Vorgaben aus GoBS und MaRisk an Softwareentwicklungsprozesse aufgelistet und deren Umsetzung im Prozess Auftragsbearbeitung betrachtet und deren Einhaltung bewertet.

Wie in Tab. 7.5 zu sehen, werden alle gesetzlichen IT-Compliance Anforderungen aus GoBS und MaRisk durch den Prozess Auftragsbearbeitung erfüllt. Dabei ist eine Berücksichtigung im Prozessmodell keine Garantie dafür, dass die Anforderungen auch tatsächlich eingehalten werden. Da der Fokus dieser Fallstudie jedoch auf der Optimierung des Software Entwicklungsprozess liegt, wird die tatsächliche Einhaltung der IT-Compliance nicht analysiert.

Tab. 7.5 Einhaltung von IT-Compliance im Prozess Auftragsbearbeitung. (Eigene Darstellung)

Anforderung (Herkunft)	*Umsetzung*
Dokumentationen müssen erstellt und mit der tatsächlich eingesetzten Version übereinstimmen (GoBS)	Die Erstellung einer fachlichen und einer technischen Dokumentation ist gemäß Prozessmodell und Ergebniskatalog verbindlich
Es müssen Richtlinien für die Programmierung, die Tests und die Freigaben geben (GoBS)	Das Prozessmodell ist die verbindliche Richtlinie für Programmierung, Tests und Freigaben
Eingesetzte Software muss autorisiert werden (GoBS)	Angepasste Software wird im Arbeitsschritt „Fachliche Freigabe erteilen (Auftrag)“ vom Auftraggeber freigegeben
Orientierung an gängigen Standards (MaRisk)	Die IT orientiert sich für viele Ihrer Prozesse an ITIL™, dem de facto Standard für IT-Service Management
Systeme müssen getestet werden (MaRisk)	Sowohl fachliche als auch technische Entwicklertests werden durch das Prozessmodell vorgegeben
Trennung von Produktions- und Testumgebungen (MaRisk)	In der DEVK sind die produktiven Systeme von den Testumgebungen getrennt
Fachliche Mitarbeiter sind zu beteiligen/ Anwender dürfen Software selber nicht freigegeben (MaRisk)	Fachliche Mitarbeiter sind im Prozess Auftragsbearbeitung an vielen Stellen involviert. Vor allem bei der Erstellung der Vorgaben und dem fachlichen Test
Gelten für jede Software (MaRisk)	Der Prozess ist für alle Software-Systeme verbindlich

[23] Vgl. MaRisk VA (2009), S. 21 f.

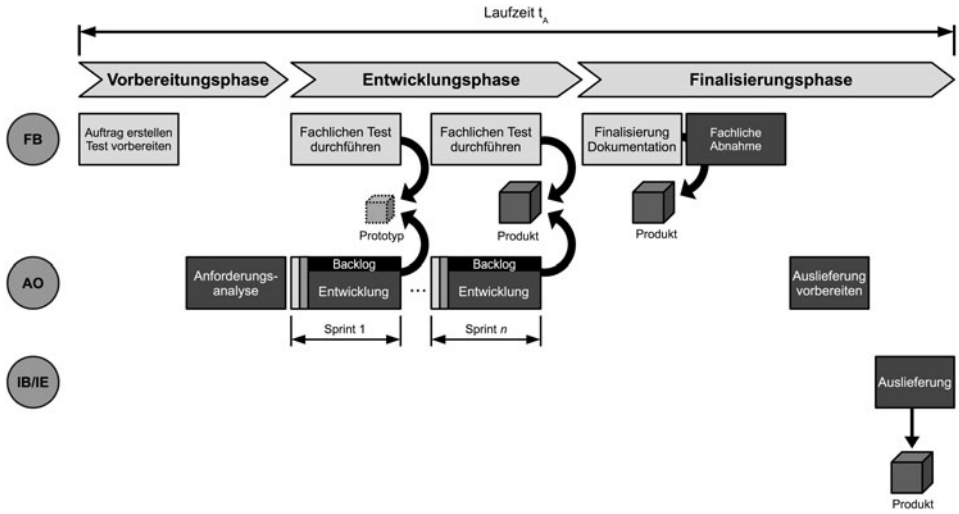

Abb. 7.2 Optimiertes Vorgehensmodell. (Eigene Darstellung)

7.4 Veränderungen durch die Einführung des hybriden Modells

In diesem Kapitel werden die Auswirkungen einer Anpassung des sequentiellen Prozesses Auftragsbearbeitung auf das hybrides Modell untersucht.

Der Teilprozess *Auftrag erstellen u. erteilen* der Auftragsbearbeitung entspricht hierbei der Vorbereitungsphase. Wenn die Teilprozesse *Fachlichen Test durchführen* und *Analyse u. Umsetzung* in Iterationen durchgeführt werden, so stellen sie die Entwicklungsphase dar. Die Finalisierungsphase wird schließlich durch die Teilprozesse *Dokumentation anpassen*, *Fachliche Abnahme durchführen*, *Auslieferung vorbereiten* und *Auslieferung durchführen* gebildet. Der so entstehende Ablauf wird in Abb. 7.2 dargestellt.

Kleine Aufträge mit lediglich einem PT Aufwand, werden nicht in Iterationen umgesetzt. Das hybride Modell wird nur einmal durchlaufen. Jedoch können mehrere kleine Aufträge ein Release bilden (siehe Kap. 5, Abschn. 5.7.1).

7.5 Erreichen der angestrebten Verbesserungen

In diesem Kapitel wird überprüft, ob die Einführung eines hybriden Modells, die gewünschten Vorteile hinsichtlich Flexibilität und Durchlaufzeit gegenüber des bestehen Prozesses leistet.

7.5.1 Verkürzung der Durchlaufzeiten

Im Sinne der agilen Softwareentwicklung sollen Mitarbeiter im hybriden Vorgehensmodell eigenverantwortlich in Zusammenarbeit mit den Fachbereichen entscheiden, welche Aufgaben sie in den nächsten Iterationen umsetzen. In der Auftragsbearbeitung wird diese Aufgabe von der Rolle *Auftragskoordinator*, die häufig von der zuständigen Führungskraft wahrgenommen wird, ausgeführt. Dadurch, dass die Aufträge nicht mehr vom Auftragskoordinator angenommen und an Mitarbeiter zugewiesen werden, reduziert sich die Durchlaufzeit in der Theorie um 4,13 Kalendertage. Sollten darüber hinaus automatisierte Akzeptanztests (siehe Kap. 5, Abschn. 5.2.6) als fachliche Vorgaben verwendet werden, lassen sich auch die durchschnittlichen Wartezeiten von 5,79 Kalendertagen vor dem fachlichen Test eliminieren, da diese jederzeit während der Umsetzung ausgeführt werden können. Nicht zu beziffern sind die Einsparungen im Bereich der Auslieferung. Durch Continuous Delivery (siehe Kap. 5, Abschn. 5.2.8) sind hier jedoch ebenfalls verkürzte Durchlaufzeiten zu erwarten.

7.5.2 Erhöhung der Flexibilität

Um zu zeigen, dass sich durch das hybride Modell die Flexibilität hinsichtlich sich ändernder Anforderungen erhöht, müssen die Prozesskosten bei einem Durchlauf mit sich ändernden Anforderungen nach der Einführung des hybriden Modells geringer sein als vorher. Durch die Einführung des hybriden Modells ergeben sich die folgenden Änderungen:

- Die Aufwände für den Teilprozess „Fachlichen Test durchführen“ werden deutlich reduziert, da sie automatisiert durchgeführt werden. In der folgenden Beispielrechnung wird angenommen, dass 20 % des ursprünglichen Aufwands (von 2,52 auf 0,5 h) erhalten bleiben, da nicht alle Aspekte einer Software automatisiert getestet werden.
- Die Aufwände für die Auslieferung werden durch Continuous Delivery ebenfalls reduziert. Da jedoch bei einer Automatisierung Fehler auftreten können, werden in der Rechnung 20 % des Aufwandes für evtl. manuelle Anpassungen beibehalten.
- Wenn sich Anforderungen ändern, wird ein Auftrag nicht mehr an den Fachbereich zurückgegeben, sondern die Akzeptanztestfälle angepasst. Daher entfällt „Auftrag erstellen“ und „entgegennehmen und prüfen“ beim zweiten Durchlauf. Dafür wird angenommen, dass der Teilprozess „Fachlichen Test vorbereiten“ zweimal durchlaufen wird, da die Akzeptanztestfälle angepasst werden.
- Durch das regelmäßige Testen während der Umsetzung werden Fehler früher entdeckt. Setzt man voraus, dass sie durchschnittlich in der der Mitte der Umsetzungszeit bemerkt werden, so ist lediglich die Hälfte des Aufwands für diesen Teilprozess und den fachlichen Test zu wiederholen. Die Prozessmenge beträgt somit in diesem Szenario für beide Teilprozesse 1,5.

Tab. 7.6 Prozesskosten des optimierten Prozesses (einfacher Durchlauf). (Eigene Darstellung)

Teilprozess	Aufwand (h)	Kostensatz (€)	Kosten/ Durchführung (€)	Prozessmenge	Kosten (€)
Auftrag erstellen u. erteilen	1,05	63,00	66,15	1	66,15
Fachlichen Test vorbereiten	1,69	63,00	106,47	1	106,47
Auftrag analysieren u. umsetzen	8,87	78,00	691,86	1	691,86
Fachlichen Test durchführen	0,5	63,00	31,50	1	31,50
Technische Dokumentation anpassen	0,97	78,00	75,66	1	75,66
Fachliche Dokumentation anpassen	1,48	63,00	93,24	1	93,24
Fachliche Abnahme durchführen	0,5	63,00	31,50	1	31,50
Auslieferung vorbereiten/ Staging	0,59	78,00	46,02	1	46,02
Staging inkl. Produktivsetzung	0,46	78,00	35,88	1	35,88
Gesamt	*16,11*		*1.178,28*	*1*	*1.178,28*

- Durch die umgestellte Reihenfolge werden einige Teilprozesse erst nach dem fachlichen Test durchgeführt und daher auch bei sich ändernden Anforderungen nur einmal durchlaufen.

Die Prozesskosten bei Verwendung des hybriden Modells für einen Durchlauf mit gleichbleibenden Anforderungen werden in Tab. 7.6 dargestellt.

In Tab. 7.7 werden die Prozesskosten für einen Durchlauf mit geänderten Anforderungen dargestellt.

Unter den genannten Voraussetzungen liegen die Prozesskosten für einen Durchlauf ohne sich ändernde Anforderungen bei 1.178,28 €. Für einen Durchlauf mit sich ändernden Anforderungen bei 1.646,43 €. Dies entspricht prozentualen Einsparungen gegenüber der Ausgangssituation in Höhe von 39,51 % bei gleichbleibenden Anforderungen und 43,29 % bei sich ändernden Anforderungen. Die Prozesskosten bei Verwendung des hybriden Mo-

Tab. 7.7 Prozesskosten des optimierten Prozesses (geänderte Anforderung). (Eigene Darstellung)

Teilprozess	Aufwand (h)	Kostensatz (€)	Kosten/ Durchführung (€)	Prozessmenge	Kosten (€)
Auftrag erstellen u. erteilen	1,05	63,00	66,15	1	66,15
Fachlichen Test vorbereiten	1,69	63,00	106,47	2	212,94
Auftrag analysieren u. umsetzen	8,87	78,00	691,86	1,5	1.037,79
Fachlichen Test durchführen	0,5	63,00	31,50	1,5	47,25
Technische Dokumentation anpassen	0,97	78,00	75,66	1	75,66
Fachliche Dokumentation anpassen	1,48	63,00	93,24	1	93,24
Fachliche Abnahme durchführen	0,5	63,00	31,50	1	31,50
Auslieferung vorbereiten/ Staging	0,59	78,00	46,02	1	46,02
Staging inkl. Produktivsetzung	0,46	78,00	35,88	1	35,88
Gesamt	*16,11*		*1.178,28*	*1*	*1.646,43*

dells liegen somit in beiden Szenarien unter den Prozesskosten des bestehenden Prozesses (siehe Kap. 7, Abschn. 7.3.2). Der Anstieg der Prozesskosten bei sich ändernden Anforderungen ist durch die Optimierung ebenfalls verringert worden. Der Aufwand steigt in diesem Fall nur noch um ca. 40 %. Vor der Optimierung waren es noch 73 %.

Diese theoretische Betrachtung zeigt, dass die Einführung eines hybriden Modells Vorteile hinsichtlich der Flexibilität bietet. Allerding muss die empirische Prüfung dieser Theorie noch erfolgen.

7.5.3 Einhaltung von IT-Compliance-Anforderungen

Die Einführung eines hybriden Vorgehensmodells bei der DEVK kann nur bei gleichzeitiger Einhaltung aller IT-Compliance-Anforderungen erfolgen.

Um dies zu überprüfen werden in Tab. 7.8 die ermittelten Anforderungen an einen Softwareentwicklungsprozess aus GoBS und MaRisk mit den geplanten Anpassungen

Tab. 7.8 Erfüllung der IT-Compliance Anforderung im optimierten Prozess. (Eigene Darstellung)

Anforderung (Herkunft)	Umsetzung
Dokumentationen müssen erstellt und mit der tatsächlich eingesetzten Version übereinstimmen (GoBS)	Die Erstellung von fachlichen und technischen Dokumentationen ist weiterhin verbindlich. Die fachliche Dokumentation wird im hybriden Modell jedoch später erstellt
Es müssen Richtlinien für die Programmierung, die Tests und die Freigaben geben (GoBS)	Wenn eine Umsetzung der erarbeiteten Maßnahmen beschlossen wird, dann sollte das Prozessmodell entsprechend aktualisiert werden. Anschließend ist die Forderung weiterhin erfüllt
Eingesetzte Software soll autorisiert werden (GoBS)	Die fachliche Freigabe ist weiterhin im Prozess enthalten
Orientierung an gängigen Standards (MaRisk)	Die Schnittstellen zu den Prozessen IT-Change- und Releasemanagement wurden nicht verändert. Damit orientieren sie sich weiterhin an ITIL™
Systeme müssen getestet werden (MaRisk)	Tests sind weiterhin verpflichtend. Bei einer Einführung von TDD und ATDD werden voraussichtlich mehr automatisierte Tests erstellt. Ein entsprechendes Protokoll kann als Testdokumentation dienen
Trennung von Produktions- und Testumgebungen (MaRisk)	Die Trennung der Produktions- und Testumgebungen bleibt von den erarbeiteten Prozessverbesserungen unberührt und ist somit weiterhin gegeben
Fachliche Mitarbeiter sind zu beteiligen/ Anwender dürfen Software selber nicht freigegeben (MaRisk)	Fachliche Mitarbeiter werden nach der Anpassung des Prozesses weiterhin eingebunden. Neu ist, dass dies kontinuierlich geschieht
Gelten für jede Software (MaRisk)	Nach einer Anpassung des Prozess-Modells ist der optimierte Prozess verbindlich und somit für jede eingesetzte Software zu nutzen

gegenübergestellt und erneut bewertet, ob die Anforderungen nach den Anpassungen weiterhin erfüllt sind.

Aus der Tab. 7.8 geht hervor, dass nach der Einführung des hybriden Modells weiterhin alle gesetzlichen Anforderungen aus den GoBS und MaRisk erfüllt werden.

7.6 Fazit

Die Fallstudie zeigt, dass die Einführung des hybriden Modells eine Möglichkeit darstellt, die Flexibilität und die Durchlaufzeit des Prozesses *Auftragsbearbeitung* zu verbessern, ohne dass die geltenden IT-Compliance-Anforderungen aus GoBS und MaRisk verletzt werden.

Allerdings ist zu beachten, dass betroffene Mitarbeiter bei der Einführung ausführlich geschult werden, da sich die vorgeschlagenen Vorgehensweisen und Techniken (z. B. Test-Driven Development und Continuous Delivery) erheblich von den bisher eingesetzten Verfahren unterscheiden.

Zusammenfassung und Ausblick

8

Viele Puristen sind der Ansicht, dass agile Methoden und klassische Vorgehensweisen nicht kombiniert werden können. So trifft zum Beispiel Cohn in seinem Buch *Agile Softwareentwicklung* die Aussage, dass er nicht an die Möglichkeit glaubt, agile und sequentielle Vorgehensweisen in einer Organisation parallel zu nutzen. Dauerhaft würden Organisationen wieder in alte Denkmuster verfallen und der notwendige KVP würde somit verhindert[1].

Seine Aussage zielt auf die unterschiedlichen Kulturen ab, die den beiden Vorgehensweisen zu Grunde liegen. Diese äußern sich bspw. in der unterschiedlichen Art Anforderungen zu erheben. Bei klassischen, phasenorientierten Vorgehensmodellen wird versucht, alle Anforderungen zu Beginn eines Vorhabens zu identifizieren. Bei agilen Methoden werden die Anforderungen hingegen eher grob erfasst und in Iterationen stetig angepasst. Gemeint sind darüber hinaus aber vor allem auch Aspekte wie die Selbstorganisation eines Teams.

Aus Sicht der Autoren lassen sich diese beiden Vorgehensweisen sehr wohl kombinieren. Agile Praktiken aus XP oder Scrum harmonieren mit einem sequenziellen Phasenmodell, das den Rahmen für ein Vorhaben bildet.

Der Vorteil dieser Kombination besteht darin, dass man durch eine grundlegende Anforderungsanalyse zu Beginn des Vorhabens in bekannter Weise einen Festpreis für die Umsetzung vereinbaren kann. Die Gefahr, dass dem zu Beginn falsche Anforderungen zu Grunde liegen oder die abgegebene Aufwandschätzung nicht zutreffend ist, bleibt auch im hybriden Modell bestehen. Durch die Iterative Entwicklung und das regelmäßige Kundenfeedback werden aber Fehlentwicklungen oder Fehlplanungen früher erkennbar. Für die üblichen Festpreisvereinbarungen kann das Modell nach heutigen Maßstäben bestmöglich genutzt werden.

[1] Vgl. Cohn (2010), S. 423.

Gregor Sandhaus et al., *Hybride Softwareentwicklung,* Xpert.press,
DOI 10.1007/978-3-642-55064-5_8

Mit dem hybriden Modell lassen sich also viele Vorteile der agilen Methoden in ein klassisches Vorgehen integrieren. Dabei muss kein drastischer Kulturwandel, den Agilisten fordern, vorgenommen werden. Vielmehr zeigt das hybride Modell Wege auf, agile Methoden sanft in einem bekannten Rahmen zu integrieren. Damit ist das hybride Modell ein Schritt zur Einführung von agilen Methoden.

Die Praxistauglichkeit des hybriden Modells wurde mit Hilfe von zwei Fallstudien über die numetris und die DEVK belegt. Das hybride Modell wird bei der numetris für Festpreisprojekte und die Pflege und Wartung des vorhandenen Produkts en|damo genutzt. Die bisherige Gestaltung des sequentiellen Prozesses wurde durch die Einführung beibehalten und die agilen Ansätze zur Unterstützung der Entwicklung und Prozessüberwachung integriert. Die agilen Ansätze ermöglichen es der Unternehmensführung und den Entwicklern, Aufwände besser abzuschätzen und frühzeitig Engpässe oder Verzögerungen zu identifizieren. Durch den Verzicht auf die Berücksichtigung spezieller Rollenverhältnisse aus Scrum und nur eine Teil-Selbständigkeit des Teams bleibt die bisherige Organisationsstruktur mit flachen Hierarchien bewahrt. Der Gedanke, den Kunden intensiver in die Entwicklung einzubinden, hat dafür gesorgt, dass Fehlentwicklungen frühzeitig aufgedeckt werden. Der zusätzliche Aufwand für die Kommunikation mit dem Kunden wird durch die ausbleibende spätere Nachbesserung aufgewogen.

Allerdings hat die Fallstudie auch gezeigt, dass agile Ansätze wie in Kap. 3, Abschn. 3.5.4 postuliert, die Wertschöpfung nicht positiv beeinflussen können. Die numetris hat sich durch die Einführung des hybriden Modells eine Optimierung der Zahlungsströme gewünscht. Es hat sich allerdings während der Einführung des hybriden Modells und der Erstellung der Studie gezeigt, dass auch bei einem Entwicklungsvorhaben nach einem klassischen Modell – zumindest auf Seiten des Auftragnehmers – eine identische Wertschöpfung erreicht werden kann. Allerdings bekommt bei einem klassisch organisierten Vorhaben der Kunde erst das Produkt am Ende des Projekts ausgeliefert, während er bei einem nach dem hybriden Modell organisierten Projekt bereits früh einen Prototypen erhält mit dem erste Tests durchgeführt und Erfahrungen gesammelt werden können. Somit hat der Kunde bereits einen entsprechenden Gegenwert zu seinen geleisteten Zahlungen und kann diesen bereits frühzeitig bilanziell aktivieren.

Am Beispiel der DEVK wurde im Rahmen einer zweiten Fallstudie gezeigt, dass das hybride Modell auch in der Software-Wartung genutzt werden kann. Gegenüber dem bestehenden Prozess *Auftragsbearbeitung*, einem klassischen, phasenorientierten Vorgehensmodell, bietet das hybride Modell eine verbesserte Durchlaufzeit und eine erhöhte Flexibilität im Umgang mit geänderten Anforderungen. Somit unterstützt das hybride Modell den Wunsch der DEVK in der Produktgestaltung schnell und flexibel auf Kundenbedürfnisse reagieren zu können.

Zusätzlich werden die geltenden IT-Compliance Anforderungen aus GoBS und MaRisk auch im hybriden Modell erfüllt.

Jedoch sollte beachtet werden, dass die Einführung des hybriden Modells zum Zeitpunkt der Veröffentlichung dieses Buches in der DEVK noch nicht durchgeführt wurde, daher

sind viele der Ergebnisse als vorläufig anzusehen. Weiterhin ist zu berücksichtigen, dass die erwarteten Ergebnisse auch tatsächlich erreicht werden. Daher wird empfohlen, dass die im Rahmen dieses Buches durchgeführten Analysen regelmäßig vorgenommen werden, um den langfristigen Erfolg zu gewährleisten.

Literatur

Abrahamsson, P., Marchesi, F., & Maurer, F. (Hrsg.). (2009). *Agile processes in software engineering and extreme programming*. Berlin: Springer.

Anderson, D. J. (2011). *Kanban. Evolutionäres Change Management für IT-Organisationen*. Heidelberg: dpunkt.verlag.

Appelo, J. (2010). *Management 3.0. Leading agile developers, developing agile leaders*. Boston: Pearson Education.

Appelo, J. (2011). *Management 3.0. Leading agile developers, developing agile leaders*. Boston: Addison-Wesley.

Avci, O. (2008). Warum entstehen in der Anforderungsanalyse Fehler? Eine Synthese empirischer Befunden der letzten 15 Jahre. In G. Herzwurm & M. Mikusz (Hrsg.), *Industrialisierung des Software-Managements* (S. 89–104). Klagenfurt: Gesellschaft für Informatik e. V. (Proceedings).

Balzert, H. (1998). *Lehrbuch der Software-Technik. Band II*. Heidelberg: Spektrum Akademischer.

Basten, D., Joosten, D., & Mellis, W. (2011). Messung des Erfolgs von IS-Entwicklungsprojekten aus Auftragnehmerperspektive. In C. Kop (Hrsg.), *32. WI-MAW-Rundbrief* (Bd. 17, S. 5–19). Klagenfurt: Gesellschaft für Informatik e. V.

Beck, K. (2003). *Test-driven development: By example*. Boston: Addison-Wesley.

Beck, K., & Andres, C. (2010). *Extreme programming explained* (2. Aufl.). Boston: Addison-Wesley.

Becker, T. (2005). *Prozesse in Produktion und Supply Chain optimieren*. Berlin: Springer.

Becker, T. (2008). *Prozesse in Produktion und Supply Chain optimieren* (2. Aufl.). Berlin: Springer.

Beims, M. (2010). *IT-Service Management in der Praxis mit ITIL™ 3. Zielfindung – Methoden – Realisierung*. München: Hanser.

Bepple, A., & Coldewey, J. (2011). Automatisierte Akzeptanztests. In R. Pichler & S. Roock (Hrsg.), *Agile Entwicklungspraktiken mit Scrum* (S. 97–110). Heidelberg: dpunkt.verlag.

Best, E., & Weth, M. (2009). *Geschäftsprozesse optimieren. Der Praxisleitfaden für erfolgreiche Reorganisationen* (3. Aufl.). Wiesbaden: Gabler.

Blum, K., & Wörsdörfer, A. (2013). Erfahrungsbericht IT-Prozessverschlankung: Mit komplexen Strukturen kurzen Prozess machen. *OBJEKTSpektrum, o. A., 1,* 52–56.

Blumberg, S., et al. (2012). IT-Projektsteuerung – eine Methodik zum Benefits-Management mit integrierter Risikobetrachtung. *Wirtschaftsinformatik & Management, o. A., 5,* 56–60.

Bon, J. van, & Jong, A. de. (2009). *IT Service Management basierend auf ITIL™ V3. Das Taschenbuch*. Zaltbommel: Van Haren.

Borgmeier, E. (2006). Mit agilem Qualitätsmanagement zum Projekterfolg. *Java Spektrum, o. A., 5,* 13–15.

Gregor Sandhaus et al., *Hybride Softwareentwicklung,* Xpert.press,
DOI 10.1007/978-3-642-55064-5

Bremen, H. S., et al. (2011). Umfrage 2011- Softwaretest in der Praxis vom 01.05.–31.05. http://www.softwaretest-umfrage.de/Vorgehensmodelle.htm. Zugegriffen: 2. Aug. 2013. (21:50 Uhr).

Bröhl, A.-P., & Dröschel, W. (1993). *Das V-Modell: Der Standard für die Softwareentwicklung mit Praxisleitfaden*. München: Oldenbourg.

Brooks, F. P. (1995). *The mythical man-month* (20. Aufl.). Boston: Addison-Wesley.

Cockburn, A. (2003). *Agile Software-Entwicklung*. Bonn: mitp.

Coenenberg, A. G., Fischer, T. M., & Günther, T. (2012). *Kostenrechnung und Kostenanalyse*. 8. Aufl. Stuttgart: Schäffer-Poeschel.

Cohn, M. (2010). *Agile Softwareentwicklung*. München: Addison-Wesley.

Coyle, S., & Conboy, K. (2009). A study of risk management in DSDM. In P. Abrahamsson, F. Marchesi, & F. Maurer (Hrsg.), *Agile processes in software engineering and extreme programming* (S. 142–147). Berlin: Springer.

Dechko, A. (2012). Agile – das Aus für den Wasserfall? http://www.computerwoche.de/2352228. Zugegriffen: 28. Juli 2013 (15:15 Uhr).

Ditze, A., Mühlbauer, S., & Stolz, P. (2013). Modellierung – agil versus traditionell. *SQMagazin, o. A., 27,* 22–24.

Eisenführ, F., & Weber, M. (2003). *Rationales Entscheiden*. (Band 4. Aufl.). Berlin: Springer.

Ekssir, M. (2013). Warum Glaubenskriege? *SQMagazin, o. A., 27,* 10–11.

Etzel, H.-J. (2007). Externe Qualitätssicherung in Organisations-/IT-Projekten. In C. Kop (Hrsg.), *23. WI-MAW-Rundbrief*. (Bd. 13, S. 37–43). Gesellschaft für Informatik e. V.

Freidank, C.-C. (2008). *Kostenrechnung. Grundlagen des innerbetrieblichen Rechnungswesens und Konzepte des Kostenmanagements* (8. Aufl.). München: Oldenbourg.

Friedrichsen, U. (2012). Dr. Hektisch und Mr. Hype. Überleben im Wirtschaftsdarwinismus. *Business Technology, o. A., 4,* 12–20.

Gärtner, M. (2013). Die Mensch-Maschine. Ausgewogene Qualitätssicherung mit aufeinander abgestimmten Testgattungen. *iX Developer, o. A., 3,* 112–116.

Gaulke, M. (2010). *Praxiswissen COBIT Val IT – Risk IT. Grundlagen und praktische Anwendung für die IT-Governance*. Heidelberg: dpunkt.verlag (Auflage).

Gleißner, W. (2003). Die Psychologie unternehmerischer Entscheidungen. *Wirtschaftspsychologie aktuell, o. A., 2,* 69–74.

Göbl, W., & Köhler, A. (2013). Agilität und Requirements-Engineering: Ergänzung oder Widerspruch? *OBJEKTSpektrum, o. A., 4,* 79–82.

Gundlach, C., & Jochem, R. (2008). *Praxishandbuch Six Sigma. Fehler vermeiden, Prozesse verbessern, Kosten senken*. Düsseldorf: Symposion.

Hansen, H. R., Neumann, G., Bea, F. X., Friedl, B., & Schweitzer, M. (Hrsg.). (2005). *Wirtschaftsinformatik 1: Grundlagen und Anwendungen* (9. Aufl.). Stuttgart: UTB.

Hansen, H. R., Neumann, G., Bea, F. X., Friedl, B., & Schweitzer, M. (Hrsg.). (2010). *Wirtschaftsinformatik 1: Grundlagen und Anwendungen* (10. Aufl.). Stuttgart: UTB.

Henkel, E., Ober, M., & Taubner, D. (2011). Erfahrungen mit Lean-Konzepten im Management von Softwareprojekten. *Informatik Spektrum, 34*(1), 60–70.

Henrich, A. (2002). *Management von Softwareprojekten*. München: Oldenbourg.

Herzwurm, G., & Mikusz, M. (Hrsg.). (2008). Industrialisierung des Software-Managements. Band P-139, Proceedings. Stuttgart: Gesellschaft für Informatik e. V.

Hibbs, C., Jewett, S., & Sullivan, M. (2008). *The art of lean software development*. Sebastopol: O'Reilly Media Inc.

Hoffmann, D. W. (2008). *Software-Qualität*. Berlin: Springer.

Hofmann, J., & Schmidt, W. (2010). *Masterkurs IT-Management* (2. Aufl.). Wiesbaden: Vieweg+Teubner.

Höppner, S., & Höhn, R. (2005). Das V-Modell, Fortsetzung einer Erfolgsstory. In C. Kop (Hrsg.), *19. WI-MAW-Rundbrief* (Bd. 11, S. 10–13). Klagenfurt: Gesellschaft für Informatik e. V.

Humphrey, W. S. (1989). *Managing the software process*. Boston: Addison-Wesley.

Hunt, A., & Thomas, D. (2003). *Der pragmatische Programmierer*. München: Hanser.

Jammernegg, W., Poiger, M., & Unterweger, A. (2009). Produktionsmanagement. In S. Kummer (Hrsg.), *Grundzüge der Beschaffung, Produktion und Logistik* (2. Aufl., S. 215–244). München: Pearson Education Deutschland.

Jórasz, W. (2008). *Kosten- und Leistungsrechnung. Lehrbuch mit Aufgaben und Lösungen* (4. Aufl.). Stuttgart: Schäffer-Poeschel.

Jungowski, D. (2013). Scrum oder Kanban? *eclipse magazin, o. A., 3,* 59–63.

Kaluza, B., & Blecker, T. (2005). *Erfolgsfaktor Flexibilität. Strategien und Konzepte für wandlungsfähige Unternehmen*. Berlin: Erich Schmidt.

Kamann, T., & Wendt, H. (2012). On the road to continuous delivery. *Java Magazin, o. A., 5,* 58–64.

Klaus, A. (2012). Stakeholder-orientierter Software Test für Geschäftsanwendungen. *Softwaretechnik-Trends, 32*(1), 8–9.

Kneuper, R. (2005). Nutzung von CMMI und Verwandten zur Lieferantenauswahl, oder: Wie kann ich die Aussagekraft angeblicher CMMI-Assessmentergebnisse überprüfen. In C. Kop (Hrsg.), *19. WI-MAW-Rundbrief* (Bd. 11, S. 14–23). Klagenfurt: Gesellschaft für Informatik e. V.

Koch, S. (2011). *Einführung in das Management von Geschäftsprozessen. Six Sigma, Kaizen und TQM*. Berlin: Springer.

Kop, C. (Hrsg.). (2004). *17. WI-MAW-Rundbrief*. Klagenfurt: Gesellschaft für Informatik e. V.

Kop, C. (Hrsg.). (2005). *19. WI-MAW-Rundbrief*. Klagenfurt: Gesellschaft für Informatik e. V.

Kop, C. (Hrsg.). (2007a). *23. WI-MAW-Rundbrief*. Klagenfurt: Gesellschaft für Informatik e. V.

Kop, C. (Hrsg.). (2007b). *24. WI-MAW-Rundbrief*. Klagenfurt: Gesellschaft für Informatik e. V.

Kop, C. (Hrsg.). (2009). *27. WI-MAW-Rundbrief*. Klagenfurt: Gesellschaft für Informatik e. V.

Kop, C. (Hrsg.). (2010a). *29. WI-MAW-Rundbrief*. Klagenfurt: Gesellschaft für Informatik e. V.

Kop, C. (Hrsg.). (2010b). *30. WI-MAW-Rundbrief*. Klagenfurt: Gesellschaft für Informatik e. V.

Kop, C. (Hrsg.). (2011a). *31. WI-MAW-Rundbrief*. Klagenfurt: Gesellschaft für Informatik e. V.

Kop, C. (Hrsg.). (2011b). *32. WI-MAW-Rundbrief*. Klagenfurt: Gesellschaft für Informatik e. V.

Kop, C. (Hrsg.). (2013). *35. WI-MAW-Rundbrief*. Klagenfurt: Gesellschaft für Informatik e. V.

Korhonen, K. (2009). Migrating defect management from waterfall to agile software development in a large-scale multi-site organization: A case study. In P. Abrahamsson, F. Marchesi, & F. Maurer (Hrsg.), *Agile processes in software engineering and extreme programming* (S. 73–82). Berlin: Springer.

Krcmar, H. (2010). *Informationsmanagement* (5. Aufl.). Berlin: Springer.

Kriescher, I., & Markgraf, J. (2011). Agiles V-Modell – ein Widerspruch? http://www.computerwoche.de/software/software-infrastruktur/1900867/. Zugegriffen: 18. Mai 2013 (13:50 Uhr).

Kütz, M. (2009). *Kennzahlen in der IT* (3. Aufl.). Heidelberg: dpunkt.verlag.

Lami, G., & Falcini, F. (2009). Is ISO/IEC 15504 applicable to agile methods? In P. Abrahamsson, F. Marchesi & F. Maurer (Hrsg.), *Agile processes in software engineering and extreme programming* (S. 130–135). Berlin: Springer.

Lange, L., et al. (2013). Akzeptanzgetriebener Entwicklungsprozess: Klassische Werkzeuge im agilen Anforderungsmanagement. *OBJEKTSpektrum, o. A., 3,* 26–30.

Laudon, K. C., P., L. J., & Schoder, D. (2010). *Wirtschaftsinformatik. Eine Einführung* (2. Aufl.). München: Pearson Education Deutschland.

Liggesmeyer, P. (2009). Software-Qualitätssicherung gestern und heute: Theorie und Erfahrung, Standards und Common Sense. In C. Kop (Hrsg.), *27. WI-MAW-Rundbrief* (Bd. 15, S. 15–23). Klagenfurt: Gesellschaft für Informatik e. V.

Linssen, O. (2010). Agile Vorgehensmodelle aus betriebswirtschaftlicher Sicht. In C. Kop (Hrsg.), *29. WI-MAW-Rundbrief* (Bd. 16, S. 44–54). Klagenfurt: Gesellschaft für Informatik e. V.

Lorenz, S. (2012). IT zwischen Kosteneffizienz und Innovationsfähigkeit. In M. Lang & M. Amberg (Hrsg.), *Dynamisches IT-Management. So steigern Sie die Agilität, Flexibilität und Innovationskraft Ihrer IT* (S. 45–46). Düsseldorf: Symposion.

Luke, M. (2012). Aus Alt wird Neu. OBJEKTSpektrum – Sonderbeilage: Collaboration Application Lifecycle Management, o. A., Nr. o. A., 29–30.

Mackert, O., Hildebrand, T., & Podbicanin, A. (2011). Von wasserfallartigen Softwareentwicklungsmodellen hin zu Lean Software Product Development. In C. Kop (Hrsg.), *31. WI-MAW-Rundbrief* (Bd. 17, S. 48–59). Klagenfurt: Gesellschaft für Informatik e. V.

Marchenko, A., Abrahamsson, P., & Ihme, T. (2009). Long-term effects of test-driven development. A case study. In P. Abrahamsson, F. Marchesi & F. Maurer (Hrsg.), *Agile processes in software engineering and extreme programming* (S. 13–22). Berlin: Springer.

McConnell, S. (2006). *Aufwandschätzung bei Softwareprojekten*. Unterschleißheim: Microsoft.

Mertens, P. (2012). Schwierigkeiten mit IT-Projekten der Öffentlichen Verwaltung – Neuere Entwicklungen. *Informatik Spektrum, 35*(6), 433–446.

Mittermeir, R. (2004). Life-Cycle übergreifendes Qualitätsmanagement. In C. Kop (Hrsg.), *17. WI-MAW-Rundbrief* (Bd. 10, S. 26–32). Klagenfurt: Gesellschaft für Informatik e. V.

Müller, K.-R. (2011). *IT-Sicherheit mit System. Integratives IT-Sicherheits-, Kontinuitäts- und Risikomanagement – Sicherheitspyramide – Standards und Practices – SOA und Softwareentwicklung* (4. Aufl.). Wiesbaden: Vieweg+Teubner.

Myerson, M. (1996). *Risk management processes for software engineering models*. Norwood: Artech House.

Niemann, K. D. (2005). *Von der Unternehmensarchitektur zur IT-Governance. Bausteine für ein wirksames IT-Management*. Wiesbaden: Vieweg.

o. V. (2004). Spiralmodell nach Boehm (1988). http://commons.wikimedia.org/wiki/File:Spiralmodel_nach_Boehm.png. Zugegriffen: 22. Aug. 2013 (21:45 Uhr).

o. V. (2011). literate programming and the rational unified process. http://cs460-thaumkid.blogspot.de/2011/03/literate-programming-and-rational.html. Zugegriffen: 16. Juni 2013 (21:45 Uhr).

Peischl, B., & Wotawa, F. (2010). Kennzahl-gestützte, kontinuierliche Qualitätssicherung im Software Engineering. In C. Kop (Hrsg.), *29. WI-MAW-Rundbrief* (Bd. 16, S. 66–78). Klagenfurt: Gesellschaft für Informatik e. V.

Peischl, B., & Wuksch, D. (2013). Kosten-/Aufwandsabschätzung bei komplexen Software Projekten als Basis moderner IT-Governance. In C. Kop (Hrsg.), *35. WI-MAW-Rundbrief* (Bd. 19, S. 49–59). Klagenfurt: Gesellschaft für Informatik e. V.

Pichler, R. (2008). *Scrum – Agiles Projektmanagement erfolgreich einsetzen*. Heidelberg: dpunkt.verlag.

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development. An agile toolkit*. Boston: Addison-Wesley.

Poppendieck, M., & Poppendieck, T. (2007). *Implementing lean software development. From concept to cash*. Boston: Pearson Education.

Poppendieck, M., & Poppendieck, T. (2010). *Leading lean software development. Results are not the point*. Boston: Pearson Education.

Quack, K. (2013). Jedes sechste wichtige Projekt scheitert. *Computerwoche, o. A., 13-14,* 36–37.

Rechberger, E., Nissl, K., & Höhn, R. (2007). Process Improvement Modelle - Anwendung im deutschsprachigen Raum. In C. Kop (Hrsg.), *24. WI-MAW-Rundbrief* (Bd. 13, S. 49–58). Klagenfurt: Gesellschaft für Informatik e. V.

Röthig, P. (2007). WiBe 4.1: Empfehlung zur Durchführung von Wirtschaftlichkeitsbetrachtungen in der Bundesverwaltung, insbesondere beim Einsatz der IT. Berlin: Bundesministerium des Inneren (92).- Fachkonzept.

Schmidt, A. (2008). *Kostenrechnung. Grundlagen der Vollkosten-, Deckungsbeitrags und Plankostenrechnung sowie des Kostenmanagements* (5. Aufl.). Stuttgart: W. Kohlhammer.

Schmietendorf, A. (2013a). Agiles Projektmanagement vs. IT-Service-Management. *SQMagazin, o. A., 27,* 14–16.

Schmietendorf, A. (2013b). Mögliche Messansätze innerhalb agil durchgeführter Softwareentwicklungsprojekte. In C. Kop (Hrsg.), *35. WI-MAW-Rundbrief* (Bd. 19, S. 60–70). Klagenfurt: Gesellschaft für Informatik e. V.

Schmietendorf, A., & Dumke, R. (2009). Fragen der Aufwandsschätzung bei agil durchgeführten Softwareentwicklungsprojekten. In C. Kop (Hrsg.), *27. WI-MAW-Rundbrief* (Bd. 15, S. 50–61). Klagenfurt: Gesellschaft für Informatik e. V.

Schmietendorf, A., & Dumke, R. (2010). Aufwandsschätzung bei Projekten im Kontext serviceorientierter Architekturen. In C. Kop (Hrsg.), *30. WI-MAW-Rundbrief* (Bd. 16, S. 75–86). Klagenfurt: Gesellschaft für Informatik e. V.

Schuh, P. (2005). *Integrating agile development in the real world*. Hingham: Charles River Media Inc.

Schwaber, K. (2004). *Agile project management with scrum*. Redmond: Microsoft.

Schweitzer, M., & Küpper, H.-U. (2008). *Systeme der Kosten- und Erlösrechnung* (9. Aufl.). München: Vahlen.

Sneed, H. M. (2004). Jenseits vom IT-Projektmanagement. In C. Kop (Hrsg.), *17. WI-MAW-Rundbrief* (Bd. 10, S. 33–44). Klagenfurt: Gesellschaft für Informatik e. V.

Sneed, H. M. (2007a). Bedeutung der Projektaufwandsschätzung für den Festpreiserfolg. In C. Kop (Hrsg.), *23. WI-MAW-Rundbrief* (Bd. 13, S. 55–64). Klagenfurt: Gesellschaft für Informatik e. V.

Sneed, H. M. (2007b). Der Function-Point hat ausgedient. In C. Kop (Hrsg.), *24. WI-MAW-Rundbrief* (Bd. 13, S. 73–82). Klagenfurt: Gesellschaft für Informatik e. V.

Sneed, H. M. (2007c). Ein Vorgehensmodell für Integrationsprojekte. In C. Kop (Hrsg.), *24. WI-MAW-Rundbrief* (Bd. 13, S. 59–72). Klagenfurt: Gesellschaft für Informatik e. V.

Sneed, H. M. (2010). Werterhaltung von Software durch evolutionäre Qualitätssicherung. In C. Kop (Hrsg.), *30. WI-MAW-Rundbrief* (Bd. 16, S. 87–103). Klagenfurt: Gesellschaft für Informatik e. V.

Sneed, H. M. (2011). Strategien für die Evolution von IT-Anwendungssystemen. In C. Kop (Hrsg.), *32. WI-MAW-Rundbrief* (Bd. 17, S. 77–87). Klagenfurt: Gesellschaft für Informatik e. V.

Sneed, H. M. (2013). Evolutionsschätzung. In C. Kop (Hrsg.), *35. WI-MAW-Rundbrief* (Bd. 19, S. 71–80). Klagenfurt: Gesellschaft für Informatik e. V.

Sommerville, I. (2007). *Software engineering* (8. Aufl.). Harlow: Addison-Wesley.

Stephens, M., & Rosenberg, D. (2003). *Extreme programming refactored: The case against XP*. New York: Apress.

Stober, T., & Hansmann, U. (2010). *Agile software development*. Berlin: Springer.

Thoma, H. (2010). Zur Software-Qualität bei der Entwicklung und dem Betrieb von Informationssystemen – ein Erfahrungsbericht. In C. Kop (Hrsg.), *29. WI-MAW-Rundbrief* (Bd. 16, S. 110–128). Klagenfurt: Gesellschaft für Informatik e. V.

Version, O. (2011). *State of agile*. Atlanta: VersionOne – Technischer Bericht.

Wieczorrek, H. W., & Mertens, P. (2008). *Management von IT-Projekten* (3. Aufl.). Berlin: Springer.

Wiegand, D. (2013). Schreib's ins Wiki. *c't, o. A., 17,* 3.

Wiest, S. (2011a). *Continuous Integration mit Hudson. Grundlagen und Praxiswissen für Einsteiger und Umsteiger*. Heidelberg: dpunkt.verlag.

Wiest, S. (2011b). *Continuous Integration mit Hudson*. München: dpunkt.

Wischki, C. (2009). *ITIL® V2, ITIL® V3 und ISO/IEC 20000. Gegenüberstellung und Praxisleitfaden für die Einführung oder den Umstieg*. München: Hanser.

Wöhe, G., & Döring, U. (2008). *Einführung in die Allgemeine Betriebswirtschaftslehre* (23. Aufl.). München: Vahlen.

Womack, J. P., & Jones, T. J. (2013). *Lean Thinking. Ballast abwerfen, Unternehmensgewinne steigern*. Frankfurt a. M.: Campus.