



Uwe Vigerschow

# APM – Agiles Projektmanagement

Anspruchsvolle Softwareprojekte  
erfolgreich steuern

→ Unter Mitarbeit von Andrea Grass, Alexandra Augstin und Michael Hofmann

**dpunkt.verlag**



**Uwe Vigneschow** ist Abteilungsleiter bei Werum IT Solutions. In das Buch sind über 25 Jahre Erfahrung in der Softwareentwicklung als Entwickler, Berater, Projektleiter und Führungskraft eingeflossen. Mit agilen Konzepten befasst er sich seit Ende der 1990er-Jahre und hat APM, Scrum und XP bei verschiedenen Firmen eingeführt und an besondere Rahmenbedingungen angepasst.



**Andrea Grass** arbeitet als Trainerin und Beraterin für die oose Innovative Informatik eG. Sie führt Agilitätschecks durch, unterstützt Teams darin, Agilität zum Leben zu erwecken und im Unternehmen zu verankern. Mit großer Begeisterung und Engagement begleitet sie Gruppen auf ihrem Weg, ein eingespieltes und lauffähiges Team zu werden.



**Alexandra Augstin** coacht agile Teams bei einem der weltweit führenden Unternehmen der Gamesbranche. Zuvor war sie lange Zeit als Trainerin und Beraterin mit den Schwerpunkten agiles Projektmanagement, Kommunikation und Change Management tätig und einige Jahre als Entwicklungsingenieurin in der Halbleiterindustrie beschäftigt.



**Dr. Michael Hofmann** ist Arbeits- und Organisationspsychologe. Seit 2010 hilft er als Trainer und Berater für die oose Innovative Informatik eG Unternehmen dabei, Lean, Agile und Scrum einzuführen.

**Uwe Vigenschow**

# **APM – Agiles Projektmanagement**

**Anspruchsvolle Softwareprojekte  
erfolgreich steuern**

Unter Mitarbeit von Andrea Grass, Alexandra Augstin und  
Michael Hofmann



**dpunkt.verlag**

Uwe Vigenschow  
uwe@vigenschow.com

Lektorat: Christa Preisendanz  
Copy Editing: Ursula Zimpfer, Herrenberg  
Satz: Uwe Vigenschow, Hamburg  
Herstellung: Frank Heidt  
Umschlaggestaltung: Helmut Kraus, [www.exclam.de](http://www.exclam.de)  
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

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

ISBN:  
Buch 978-3-86490-211-6  
PDF 978-3-86491-631-1  
ePub 978-3-86491-632-8

1. Auflage 2015  
Copyright © 2015 dpunkt.verlag GmbH  
Wiebinger Weg 17  
69123 Heidelberg

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.

Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor noch Verlag können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.

5 4 3 2 1 0

---

# Vorwort

Warum und wann ist es für Sie sinnvoll, sich mit *APM* zu befassen? Darauf gibt es eine kurze, plakative Antwort, wie sich *APM* in die Welt der agilen Projektmanagementvorgehensweisen einordnet:

Wenn Sie Scrum machen können, machen Sie Scrum! Wenn Ihnen Scrum als agiles Framework zu umfangreich erscheint, setzen Sie Kanban ein! Wenn Sie jedoch aus unterschiedlichen Gründen mehr brauchen als Scrum, dann ist *APM* die richtige Methode für Sie!

Was bedeutet das? *APM* steht für ein **Agiles Projektmanagement** von anspruchsvollen Softwareprojekten. *APM* kann wesentlich umfassender eingesetzt werden, als es Scrum in seiner reinen Form ermöglicht. Sie erhalten in diesem Buch einen praxisorientierten Überblick über *APM* und seinen effizienten und erfolgreichen Einsatz in der Projektarbeit. Sie erfahren, wie von der Projektvorbereitung bis zu einem agilen Requirements Engineering und einer durchgängigen Softwarearchitektur agil entwickelt werden kann. Dabei skaliert *APM* von Sieben-Personen-Teams bis hin zu Großprojekten mit mehreren 100 Mitarbeitern in verteilten Teams. *APM* wird auch für die Softwareentwicklung in regulierten Umfeldern wie der Luftfahrt- und Automotive-Industrie oder der Medizin- oder Pharmatechnik eingesetzt, bei denen besonders hohe Sicherheitsvorgaben einzuhalten sind. *APM* kann auf Ihre spezifischen Anforderungen angepasst werden und ermöglicht Lösungswege, die in der Praxis funktionieren. Hinter dem aktuellen Stand von *APM* stecken mittlerweile über zehn Jahre Erfahrung.

*APM* beschreibt eine Methodik für agile Projekte, die erstmals 2006 von Bernd Oestereich und Christian Weiss veröffentlicht wurde [118]. Seitdem haben wir *APM* kontinuierlich weiterentwickelt, was letztendlich zu diesem aktuellen Buch zur *APM*-Methodik geführt hat. Bernd und Christian haben sich in den letzten Jahren anderen Aufgaben gewidmet, sodass ich als ein Wegbegleiter seit der ersten Stunde diese Aufgabe gerne übernommen habe.

## Via negativa

Michelangelo soll auf die Frage, wie er seine meisterhafte Skulptur des David erschaffen habe, geantwortet haben, dass er einfach nur alles ent-

fernt habe, was nicht nach David aussah. Man bezeichnet dieses aus der Theologie entlehnte Vorgehen als *Via negativa*, also den Weg des Verzichts und unbedingten Reduzierens auf das Minimum [45]. Das bereits den alten Griechen und Römern bekannte Prinzip beschreibt für mich sehr gut, was agiles Projektmanagement bedeutet. Ohne genau zu wissen, wie wir in einer bestimmten Situation optimal agieren, können wir zumindest mit unserem Vorgehen schädliches Handeln vermeiden. Wir besinnen uns auf die agilen Werte, Prinzipien und Praktiken.

Das klingt überschaubar. Dennoch ist die Umsetzung in der Praxis nicht trivial. *APM* unterstützt Sie dabei, die relevanten Techniken einzusetzen. Somit bietet *APM* einen gut gefüllten Werkzeugkasten für viele unterschiedliche Situationen im agilen Projektmanagement.

### **Begriffe und geschlechtsspezifische Bezeichnungen**

In der globalen Softwareentwicklung dominiert eine englische Begriffsbildung. Diesem Fakt tragen wir Rechnung. Daher werden geläufige Begriffe in *APM* entsprechend auf Englisch benutzt. Die Bezeichnung von Rollen erfolgt wie in der Praxis oft nur in der männlichen Form, obwohl natürlich Frauen ebenso alle *APM*-Rollen erfolgreich ausfüllen.

### **Danksagung**

Mein Dank gilt zahlreichen Personen. In alphabetischer Reihenfolge nach Vornamen sind dies: Bernd Oestereich, Carsten Bierans, Christian Weiss, Guido Zockoll, Heinrich Hambloch, Henning Landgrebe, Jan Gentsch, Jan Wegener, Klaas Reineke, Marcel Ecks, Marcus Winteroll, Markus Wittwer, Matthias Ferdinand, Nils Aue, Nils Bokholt, Oliver Busch, Oliver F. Lehmann, Stefan Toth, Stefan Zörner und Stephan Roth.

Mit Andrea Grass und Alexandra Augstin habe ich *APM* während unserer gemeinsamen Zeit bei oose intensiv weiterentwickelt. Michael Hofmann hat mich bei Kapitel 3 maßgeblich unterstützt und das Kapitel 23 geschrieben. Dafür gebührt den Dreien mein großer Dank! Ganz besonders möchte ich auch den Projektleitern, Product Ownern, Scrum Mastern, Entwicklern, Architekten und Testern, die ich in ihren Projekten beraten durfte, danken. Ines Meyrose danke ich für die Skizzen und Zeichnungen, die ich in den Abbildungen verwenden durfte.

Das Buch wäre nichts geworden ohne die tolle Unterstützung durch den dpunkt.verlag und Christa Preisendanz sowie Ursula Zimpfer. Insbesondere danke ich Prof. Dr. Heidi Heilmann und den anderen Reviewern des Manuskripts für ihre wertvollen und konstruktiven Anregungen.

Uwe Vigenschow, Hamburg, im Dezember 2014

---

# Aufbau und Überblick

Das Projektmanagement-Framework *APM* bietet umfangreiche Praktiken, die in diesem Buch beschrieben werden. Als Navigation und Orientierung finden Sie hier eine Übersicht über die Struktur des Buchs mit seinen fünf Teilen und insgesamt 23 Kapiteln.

## Teil I – Agilität und *APM*-Einführung

- 1 Die Architektur von *APM*** Hier lernen Sie das Fundament und die fünf Säulen von *APM* kennen, und wir führen einige zentrale Begriffe ein, die für das Verständnis von *APM* essenziell sind. Diese Begriffe werden später noch genauer definiert. Dieses Grundverständnis benötigen wir für die ersten Darstellungen der Abläufe in agilen Projekten.
- 2 Was ist Agilität?** Es gibt unterschiedliche Sichtweisen auf Agilität sowie agile Werte und Prinzipien und darauf, was das in der Praxis bedeutet. Wir werfen einen Blick auf die *APM*-Sichtweise und die systemtheoretischen Hintergründe agiler Vorgehensweisen.
- 3 Die wirtschaftliche Sicht** Letztendlich zählt der wirtschaftliche Erfolg. Ein Projekt muss sich rechnen. An diesem Punkt hat sich jeder Managementansatz zu bewähren. Und genau dort können agile Ansätze Erfolg versprechend wirken. Mit *APM* steht Ihnen eine Möglichkeit bereit, Agilität auch in größeren Projekten bzw. im Umfeld großer Organisationen zu etablieren und gleichzeitig wirtschaftlich erfolgreich zu sein.
- 4 Was kann Agilität leisten?** Mit Agilität erreichen wir nachweislich einen schneller nutzbaren Stand eines Softwareprodukts und wir reduzieren aktiv Risiken in der Softwareentwicklung, wodurch u. a. die Erfolgswahrscheinlichkeit erhöht wird. Hier blicken wir hinter die offensichtlichen Aussagen über Agilität und betrachten den individuellen Verbesserungsprozess und die Flexibilität, die über die Umsetzung verschiedener Konzepte gewonnen wird, ebenso wie die Risiken, die mit einer agilen Vorgehensweise verbunden sind.
- 5 Die Konzepte und Methoden von *APM*** Sie lernen in diesem zentralen Kapitel des ersten Teils die Konzepte und Methoden in sich geschlossen im Zusammenhang kennen. Wir betrachten die Struktur und den

Umgang mit Anforderungen, die notwendigen Artefakte und die grundlegenden Meetings mit Projektbeteiligten und/oder Stakeholdern zur Sicherstellung der internen und externen Informationsflüsse und der Rückkopplungsschleifen, die essenziell sind für den wirkungsvollen Einsatz von *APM* und somit für den Projekterfolg.

- 6 **Das Rollenmodell** *APM* besitzt ein skalierbares und flexibles Rollenmodell, um unterschiedlich große Projekte adressieren zu können. Es gibt dafür ein einfaches Rollengrundmodell und verschiedene Ausbaustufen für unterschiedliche Projektgrößen und Rahmenbedingungen.
- 7 **Das Phasenmodell** In *APM* werden zwei Varianten eines Phasenmodells unterschieden, um Neuentwicklungen und Wartungsprojekte angemessen abbilden zu können. Über diese Phasen wird eine zeitliche Einteilung und Struktur des *APM*-Projekts ermöglicht.

## Teil II – Das Projekt aufsetzen

- 8 **Ein agiles Projekt vorbereiten** In diesem Kapitel erfahren Sie, was alles zur Vorbereitung eines agilen Projekts gehört. Sie lernen außerdem das Fallbeispiel kennen, auf das wir in den folgenden Kapiteln immer wieder zurückkommen, um einzelne Aspekte zu illustrieren.
- 9 **Die Projektvorbereitung im Detail** Wir gehen durch die Artefakte von der Vision über die erste Releaseplanung bis zum agilen Projektstrukturplan. Hinzu kommen ein erster Blick auf das Risikomanagement und die Durchführung eines Projekt-Kick-off.
- 10 **Umfang und Aufwand schätzen** Die Prinzipien und Techniken der agilen Schätzweise werden erläutert und die zugrunde liegenden Konzepte aus Teil I weiter verfeinert. Konkrete Techniken zur Vorab-Projekt-schätzung werden ebenso beschrieben wie die Ursachen für die Schwierigkeiten dabei und das Einplanen von Risikoreserven.

Hier gehen wir primär auf die initiale Schätzung ein. In agilen Projekten ist *Planung* ein regelmäßiger, beinahe schon kontinuierlicher Vorgang, der uns in jeder Iteration begleitet. Planung basiert auf Schätzungen. Damit gehören Schätzmethode zur Projektmanagementbasis. Mit dem iterativen Schätzen beschäftigen wir uns weiter in Kapitel 15.

- 11 **Agile Projekte und Verträge** Agile Projekte legen den Fokus auf hohe gegenseitige Transparenz und enge, direkte Zusammenarbeit von Auftraggeber und Auftragnehmer. Das schlägt sich in der Gestaltung von Verträgen nieder und wird hier mit allen seinen möglichen Konsequenzen beschrieben. Zusätzlich ergeben sich diverse rechtliche Aspekte, für deren Klärung Sie bitte einen Experten hinzuziehen, bevor Sie einen ersten Vertrag für ein agiles Projekt abschließen!

## Teil III – Agile Architektur

- 12 **Softwarearchitektur** Was bedeutet Softwarearchitektur für agile Projekte? Der Begriff *Architektur* wird in diesem Sinne geschärft und in den Kontext agiler Projekte gestellt. Dazu wird das Modell der *Architektur-Brezel* als Prozess- und Entscheidungsmodell erläutert. *APM* zeichnet sich durch regelmäßige Workshops zur Bewertung der Architektur aus, um technische Schulden auf der Architekturebene über das Mittel der *Tech-Story* zeitnah zu tilgen. Des Weiteren werden Techniken für gemeinsame Architekturentscheidungen vorgestellt und der Bezug zwischen Architektur und Qualität über Szenarien hergestellt.
- 13 **Der Architekt und die Architekturphase** Die Rolle des Architekten in einem agilen Projekt als *Architecture Owner* wird im Detail beschrieben. Er schafft einen Rahmen, um die Mitglieder der Teams direkt in die Entscheidungsfindung einzubinden, und treibt die Architekturthemen aktiv voran. Damit bedarf es für die Rolle des Architecture Owner sowohl der Fähigkeiten eines technischen Experten als auch eines Moderators und Organisators.

## Teil IV – Konstruktion und Releases

- 14 **Struktur und Dynamik einer Iteration** In der Konstruktionsphase entfalten agile Vorgehensweisen vollends ihre Stärken und Möglichkeiten. Die Iterationen laufen nach einem festen Schema ab und wir erstellen eine Abfolge nutzbarer Releases. Die innere Struktur einer Iteration wird mit ihren Aktivitäten beschrieben. Die Planung einer Iteration wird ebenso detailliert erläutert wie der Entwicklungsmikroprozess und die testgetriebene Entwicklung. Abschließend wird die Durchführung von Retrospektiven am Ende jeder Iteration und ihr Wert für die kontinuierliche Prozessverbesserung dargestellt.
- 15 **Fortlaufende Backlog-Arbeit** Wir gehen in diesem Kapitel auf die fortlaufende und hierarchische Zerlegung eines Epos bis zur verfeinerten User Story über eine Reihe von *Splitting Patterns* ein. Auch die Themen agiles Schätzen, Refactoring und Planungsreserven werden wieder aufgegriffen und eingehend betrachtet.
- 16 **Regelmäßige hochwertige Releases** Das Schneiden von Releases sowie die Unterscheidung zwischen internen und externen Releases und deren Planung werden im Detail beschrieben. In diesem Zusammenhang gehen wir auch auf Qualitätsaspekte ein.
- 17 **Projektcontrolling und -steuerung** Das auf der Velocity basierende agile Projektcontrolling wird mit seinen Visualisierungen beschrieben. Dabei geht es um die Steuerung des Projekts und der Aktivitäten innerhalb ei-

ner Iteration. Metriken für agile Projekte und ihr Nutzen bei der Selbstorganisation und -steuerung der Teams werden erläutert.

- 18 **Kanban und Lean Management** Hier werden Kanban und Lean Management als eine der fünf Säulen von *APM* genauer beschrieben. Der Einsatz dieser Methoden hilft besonders bei der Ergebnisorientierung und der Früherkennung negativer Trends in der Projektumsetzung.
- 19 **Der agile Coach im täglichen Einsatz** Die vielfältigen und zahlreichen Aufgaben eines agilen Coaches werden beleuchtet und das Konzept der adaptiven Führung erläutert. Wie trägt ein agiler Coach zur Wertschöpfung bei? Welche Anforderungen werden an diese Rolle geknüpft? Was bedeuten Coaching, Mentoring und Moderation genau? Auch auf die agilen Werte und wie sie zum Leben erweckt werden können, wird in diesem Zusammenhang eingegangen.

## Teil V – Agile Großprojekte

- 20 **APM für große Projekte skalieren** *APM* kann als vorskaliertes agiles Projektmanagement-Framework für mittelgroße Softwareprojekte mit zwei bis fünf parallel arbeitende Featureteams eingesetzt werden. In diesem Kapitel betrachten wir die Konzepte, um *APM* zu skalieren. Das Scrum-of-Scrums-Konzept wird auf *APM* übertragen und die Konsequenzen für die Featureteams und deren Zusammenarbeit sowie für das Projektleitungsteam werden analysiert. Die interne Struktur einer Iteration mit Fortschritts- und Orientierungsteil spielt dabei eine besondere Rolle für die Kommunikation in Großprojekten ebenso wie die Communities of Practice sowie der Einsatz von Persona und Szenarien.
- 21 **Verteilte Teams** Zuerst analysieren wir verschiedene Arten der Verteilung von Teams und betrachten mögliche Risiken, die sich daraus ergeben. Vorgehensweisen und Strategien werden erläutert und wie sie sich einzeln oder kombiniert umsetzen lassen.
- 22 **Besonderheiten im regulierten Umfeld** Erprobte agile Vorgehensweisen sind mittlerweile auch bei der Softwareentwicklung unter besonders hohen Sicherheitsvorgaben üblich und werden von den entsprechenden Richtlinien unterstützt. Wir betrachten exemplarisch an GAMP 5 und AAMI TIR45 die Konsequenzen, die sich daraus ergeben, und ergänzen die risikoorientierte Vorgehensweise in *APM* um entsprechende Vorgehensweisen.
- 23 **Agilität im Unternehmen einführen** Zum Abschluss betrachten wir die Einführung von Agilität in einem mittelständischen oder großen Unternehmen. Die Einführung von agilem Projektmanagement ist ein Change-Management-Projekt und die entsprechende Organisationsentwicklung ist nach agilen Prinzipien zu gestalten.

# Inhaltsverzeichnis

<b>I</b>	<b>Agilität und APM-Einführung</b>	<b>1</b>
<b>1</b>	<b>Die Architektur von APM</b>	<b>3</b>
1.1	Was ist <i>APM</i> ?	3
1.2	Ein paar grundlegende Begriffe vorab	6
<b>2</b>	<b>Was ist Agilität?</b>	<b>9</b>
2.1	Etwas Systemtheorie zur Einstimmung	9
2.2	Agile Werte, Prinzipien und Praktiken	15
<b>3</b>	<b>Die wirtschaftliche Sicht</b>	<b>25</b>
3.1	Nur implementieren, was notwendig ist	25
3.2	Einen schnellen Return on Investment forcieren	26
<b>4</b>	<b>Was kann Agilität leisten?</b>	<b>31</b>
4.1	Die richtigen Fragen aufwerfen	31
4.2	Ein kontinuierlicher Verbesserungsprozess	33
4.3	Flexible Lösungen – Satisficing	34
4.4	Risiken von Agilität	36
<b>5</b>	<b>Die Konzepte und Methoden von APM</b>	<b>41</b>
5.1	Vertiefung der Definitionen	41
5.2	Anforderungen – User Stories und Story Map	46
5.3	Artefakte	53
5.4	Meetings	63
<b>6</b>	<b>Das Rollenmodell</b>	<b>73</b>
6.1	Das Rollenmodell nach Scrum	73
6.2	Mögliche Rollen in <i>APM</i>	75
6.3	Trotz differenzierter Rollen agil bleiben	81
<b>7</b>	<b>Das Phasenmodell</b>	<b>85</b>
7.1	Traditionelle Phasenmodelle	85
7.2	Agile Phasenmodelle	89
7.3	Das <i>APM</i> -Phasenmodell	91

<b>II Das Projekt aufsetzen</b>		<b>97</b>
<b>8</b>	<b>Ein agiles Projekt vorbereiten</b> .....	<b>99</b>
8.1	Inhalte der Vorbereitungsphase .....	99
8.2	Wer bereitet das Projekt vor? .....	101
8.3	MyHeavyShop – unser Fallbeispiel .....	103
<b>9</b>	<b>Die Projektvorbereitung im Detail</b> .....	<b>105</b>
9.1	Produktvision und Systemkontext .....	105
	<i>Infobox: Produktvision MyHeavyShop</i> .....	107
9.2	Externe Stakeholder und Kommunikationsplan .....	108
9.3	Produkt-Roadmap und Releaseplanung .....	112
9.4	Agiler Projektstrukturplan .....	116
9.5	Risikomanagement .....	119
9.6	Projekt-Kick-off durchführen .....	128
<b>10</b>	<b>Umfang und Aufwand schätzen</b> .....	<b>131</b>
10.1	Investitionen, Budgets und Nutzen .....	131
10.2	Agile Aufwandsschätzung .....	135
10.3	»Ideale Tage« oder Komplexitätspunkte? .....	140
10.4	Velocity – Wann sind wir fertig? .....	142
10.5	Grobe Projektschätzung – Kosten und Dauer .....	144
10.6	Risikoreserven vorsehen .....	152
<b>11</b>	<b>Agile Projekte und Verträge</b> .....	<b>155</b>
11.1	Kernstück eines Vertrags für agile Projekte .....	155
11.2	Kostenfaktoren ermitteln .....	156
11.3	Vertragsgestaltung für agile Projekte .....	161
<b>III Agile Architektur</b>		<b>165</b>
<b>12</b>	<b>Softwarearchitektur</b> .....	<b>167</b>
12.1	Was ist eigentlich Softwarearchitektur? .....	167
12.2	Architektur und Agilität .....	171
	<i>Infobox: Prioritäten finden durch Punkteleben</i> .....	176
12.3	Entscheidungen treffen .....	177
12.4	Architektur und Qualität .....	182
<b>13</b>	<b>Der Architekt und die Architekturphase</b> .....	<b>189</b>
13.1	Der agile Architekt als Architecture Owner .....	189
13.2	Die Aufgaben in der Architekturphase .....	192

<b>IV</b>	<b>Konstruktion und Releases</b>	<b>197</b>
<b>14</b>	<b>Struktur und Dynamik einer Iteration</b>	<b>199</b>
14.1	Schema einer Iteration	199
14.2	Die Iteration planen	204
14.3	Planning Poker	206
14.4	Mikroprozess und testgetriebene Entwicklung	209
14.5	Retrospektiven durchführen	215
<b>15</b>	<b>Fortlaufende Backlog-Arbeit</b>	<b>219</b>
15.1	Timeboxing – Ziele erreichen	219
15.2	Vom Epos zur User Story	226
15.3	Anforderungen zerlegen – Splitting Pattern	237
15.4	Refactoring im Großen wie im Kleinen	243
	<i>Infobox: Zerbrochene Fenster und Null-Toleranz</i>	246
15.5	Mit Reserven umgehen	251
<b>16</b>	<b>Regelmäßige hochwertige Releases</b>	<b>257</b>
16.1	Die Macht des Rhythmus	257
16.2	Releases schneiden und erstellen	261
	<i>Infobox: Relatives Schätzen mit T-Shirt-Größen</i>	264
16.3	Produktqualität als permanentes Ziel	266
<b>17</b>	<b>Projektcontrolling und -steuerung</b>	<b>275</b>
17.1	Grundprinzipien des agilen Projektcontrollings	275
17.2	Projektcontrolling – Story Points und Tasks	280
17.3	Eine Iteration über das Taskboard steuern	287
17.4	Metriken und KPI	288
17.5	Metriken für agile Projekte	292
	<i>Infobox: Der Wert-Begriff in agilen Metriken</i>	294
17.6	»Caves and Common« und Selbststeuerung	303
	<i>Infobox: Die negative Wirkung von Lärm auf die Produktivität</i>	305
17.7	Anforderungsänderungen willkommen heißen	309
<b>18</b>	<b>Kanban und Lean Management</b>	<b>313</b>
18.1	Kanban kurz und knapp	313
18.2	Lean Management	315
18.3	Die drei Kanban-Prinzipien	319
18.4	Mit Kanban steuern	323
18.5	Multitasking bei Projekten vermeiden	325
18.6	Value Stream Map	326
18.7	Kanban in der Entwicklung nutzen	329

<b>19</b>	<b>Der agile Coach im täglichen Einsatz</b>	<b>333</b>
19.1	Aufgaben eines agilen Coaches	333
	<i>Infobox: GROW – Coaching als Prozess</i>	337
19.2	Teams aufbauen	342
19.3	Anforderungen an einen agilen Coach	346
19.4	Führung in agilen Projekten	349
19.5	Die agilen Werte zum Leben bringen	353

**V Agile Großprojekte** **361**

<b>20</b>	<b>APM für große Projekte skalieren</b>	<b>363</b>
20.1	Featureteams und Teamprojektleiterstruktur	363
20.2	Fortschritt und Orientierung in einer Iteration	373
20.3	Community of Practice	382
20.4	Arbeiten mit Persona-Beschreibungen	386
<b>21</b>	<b>Verteilte Teams</b>	<b>393</b>
21.1	Herausforderungen und Konsequenzen	393
21.2	Vorgehensweisen und Strategien	398
<b>22</b>	<b>Besonderheiten im regulierten Umfeld</b>	<b>403</b>
22.1	FDA, GMP und das ganze Alphabet	403
22.2	Traceability – Nachvollziehbarkeit herstellen	407
22.3	Risikobasiertes Entwickeln und Testen	409
22.4	Dokumentenreviews? Pair Working!	415
22.5	Besonderheiten bei automatisierten Tests	416
22.6	Die Teamstärke und Rollen anpassen	417
<b>23</b>	<b>Agilität im Unternehmen einführen</b>	<b>419</b>
23.1	Organisationsentwicklung auf zwei Ebenen	419
23.2	Die Verbreitung auf der operativer Ebene	421
	<b>Abschlussbemerkung und Literaturtipps</b>	<b>427</b>
	<b>Referenzen und weiterführende Literatur</b>	<b>431</b>
	<b>Index</b>	<b>443</b>

# Teil I

## Agilität und APM-Einführung

---

### 1 Die Architektur von APM 3

Hier lernen Sie das Fundament und die fünf Säulen von *APM* kennen, und wir definieren einige zentrale Begriffe, die für das Verständnis von *APM* essenziell sind.

### 2 Was ist Agilität? 9

Die systemtheoretischen Hintergründe agilen Vorgehens beleuchten wir hier ebenso wie die Werte und Prinzipien agiler Vorgehensweisen.

### 3 Die wirtschaftliche Sicht 25

Letztendlich muss sich ein Projekt rechnen. Gerade hier bieten agile Vorgehensweisen wie *APM* besondere Vorteile.

### 4 Was kann Agilität leisten? 31

Hier blicken wir hinter die offensichtlichen Aussagen über Agilität und betrachten den individuellen Verbesserungsprozess und die Flexibilität, die über die Umsetzung agiler Konzepte gewonnen wird, ebenso wie die Risiken, die mit einer agilen Vorgehensweise verbunden sind.

### 5 Die Konzepte und Methoden von APM 41

In diesem zentralen Kapitel des ersten Teils lernen Sie die Konzepte und Methoden von *APM* in sich geschlossen im Zusammenhang kennen.

### 6 Das Rollenmodell 73

*APM* besitzt ein skalierbares und flexibles Rollenmodell, um unterschiedlich große Projekte unter verschiedenen Rahmenbedingungen adressieren zu können.

### 7 Das Phasenmodell 85

In *APM* werden zwei Varianten eines Phasenmodells unterschieden, um Neuentwicklungen und Wartungsprojekte angemessen abbilden zu können.

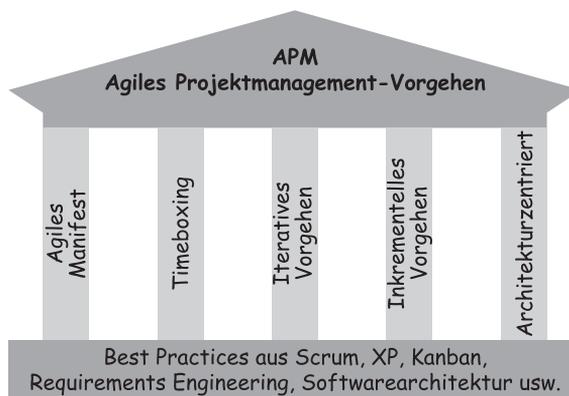


# 1 Die Architektur von APM

Agile Vorgehensweisen sind im grundsätzlichen Ansatz einfach zu verstehen, jedoch wird die innere Komplexität ihrer Umsetzung sofort deutlich, wenn wir versuchen, detailliert zu erklären, wie ein agil durchgeführtes Projekt konkret abläuft. Bevor wir in die Tiefen von *APM* eintauchen, finden Sie hier einen kurzen Überblick über die zentralen Elemente von *APM*. Danach klären wir Begriffe aus dem agilen Projektmanagement, die wir später noch genauer definieren werden, aber bereits für die ersten Darstellungen der Zusammenhänge in ihrer grundlegenden Bedeutung benötigen.

## 1.1 Was ist APM?

*APM* steht für **A**giles **P**rojekt**m**anagement und beschreibt ein Framework, um Softwareprojekte mit der notwendigen Flexibilität umzusetzen und der hohen Dynamik des Projektumfelds angemessen begegnen zu können. Es fußt auf fünf Säulen und nutzt eine Vielzahl einzelner Best Practices aus unterschiedlichen methodischen Vorgehensweisen (Abb. 1-1):



**Abbildung 1-1:** Das Fundament und die fünf Säulen von *APM*

- *APM* orientiert sich am Agilen Manifest, also an den beschriebenen vier Wertepaaren, und fußt auf den dort genannten zwölf Prinzipien als grundlegende Basis für das Vorgehen, die Führung und die Zusammenarbeit innerhalb des Projekts sowie mit den Stakeholdern [12].
- *APM* implementiert ein durchgängiges Timeboxing zur Planung und Steuerung von Projekten. Eine *Timebox* ist ein fester, vorab definierter Zeitabschnitt, in dem wir planen, ein inhaltliches Ergebnis zu erreichen, das wir am Ende überprüfen können. Dieses Konzept durchdringt *APM* von kleinen Abschnitten der täglichen Zusammenarbeit wie in täglichen kurzen Meetings über feste Zeitabschnitte von wenigen Wochen, Iterationen genannt, bis auf die Ebene der Auslieferungen.
- *APM* ist ein iteratives Vorgehen, d. h., der zeitliche Ablauf ist in gleich lange Iterationen von wenigen Wochen unterteilt, was einem zeitlichen Prüfraster mit festem Rhythmus entspricht.
- *APM* beinhaltet ein inkrementelles Vorgehen. In einer Iteration erarbeiten wir ein prüfbares Ergebnis von direktem Nutzen für den Kunden bzw. die Anwender, also ein Stück funktionierender Software. Das Ergebnis eines Projekts wird schrittweise erarbeitet. Ein Zwischenergebnis bezeichnen wir als Inkrement. Die Inkremente bauen aufeinander auf, sodass wir uns mit jedem weiteren Inkrement dem Projektergebnis konkret nähern. Damit ein inkrementell-iteratives Vorgehen funktioniert, strukturieren wir unser angestrebtes Projektergebnis derart, dass wir ausreichend kleine, sinnvolle Teilstücke als Inkremente innerhalb einer Iteration umsetzen können.
- *APM* ist ein architekturzentriertes Vorgehen. Die Arbeit an der Softwarearchitektur findet kontinuierlich statt, wobei regelmäßig im Projektverlauf grundlegende Architekturfragen im Vordergrund stehen.

Diese fünf zentralen Säulen von *APM* sind eingebettet in eine Vielzahl von Best Practices aus unterschiedlichen Frameworks und Methoden. Im Wesentlichen sind das Scrum, eXtreme Programming (XP), Kanban, agiles Requirements Engineering und Softwarearchitektur (Abb. 1-1). In *APM* wenden wir nicht nur neueste Techniken an, sondern nutzen die Erfahrung aus mehreren Jahrzehnten mit agilem Projektmanagement, Lean Management und inkrementell-iterativen Vorgehensweisen.

Damit ist *APM* besonders dafür geeignet, komplexe Projekte mit einem oder auch mehreren parallel arbeitenden Teams durchzuführen, in denen eine besonders hohe innere und äußere Qualität der Softwareprodukte gefordert ist. Ein wesentlicher Erfolgsfaktor liegt darin, dass die konkrete Vorgehensweise und die Ausprägung von *APM* im laufenden Projekt regelmäßig betrachtet und verbessert werden. Erfahrungen aus dem laufenden Projekt wirken so direkt auf das Projekt selbst und können sofort genutzt sowie den Projektteams kommuniziert werden.

*APM* stellt ein flexibles Rollenmodell bereit, über das wir weitgehend unabhängig vom Spezialisierungsgrad der einzelnen Teammitglieder cross-funktionale *Featureteams* bilden können, die jeweils gemeinsam verantwortlich einen eigenständig nutzbaren Teil des Softwareprodukts durchgängig und vollständig umsetzen [99]. So entkoppeln wir in *APM* die einzelnen Teams voneinander, sodass gegenseitig induzierte Wartezeiten zwischen den Teams minimiert werden und sich ein kontinuierlicher Projektfortschritt einstellt.

*APM* kann man auch als umfangreichen Werkzeugkasten um einen im Kern an Scrum angelehnten agilen Ablauf verstehen. Der Werkzeugkasten ist randvoll gefüllt mit zueinander kompatiblen, sich ergänzenden bzw. aufeinander aufbauenden Techniken, aus denen Sie sich zur Lösung bestimmter Aufgaben, Probleme oder Fragestellungen bedienen können. Die einzelnen Zutaten stammen ihrerseits aus unterschiedlichen Ansätzen. So finden sich z. B. auch traditionelle Techniken z. B. in der Projektvorbereitung oder im Risikomanagement wieder. Was zueinander passt, sich ergänzt, funktioniert und nicht den agilen Werten widerspricht, findet sich in *APM* wieder.

Daher passt *APM* erfahrungsgemäß besonders gut in Umfelder mit einer breiten Erfahrung in traditionellem Projektmanagement oder wo zusätzliche Anforderungen wie gesetzliche Auflagen oder andere Regularien zu erfüllen sind. Wenn Sie mit Scrum und Kanban an Grenzen gestoßen sind, weil die laufende Entwicklung stark gewachsen und an Komplexität zugenommen hat und Sie dafür eine Reihe ergänzender Praktiken etablieren möchten, kann Ihnen *APM* ebenfalls weiterhelfen.

Das offene Werkzeugkastenkonzept spiegelt auch die aktuelle Realität wider, in der die Gräben zwischen der agilen und der traditionellen Projektwelt langsam zugeschüttet werden und beeinträchtigende Dogmen beiseite geschoben werden. Wir können nur voneinander lernen.

Auch wenn *APM* auf den ersten Blick durch seinen umfangreichen Werkzeugkasten besticht, so ist es doch wesentlich mehr. *APM* basiert komplett auf der Umsetzung der agilen Werte und Prinzipien aus dem Agilen Manifest. *APM* ist kein traditionelles Vorgehen, das um ein paar Techniken aus Scrum erweitert wurde. Es ist ein durchweg agiles Vorgehen, bei dem vorurteilsfrei geschaut wird, was bei bestimmten Fragestellungen hilfreich sein kann. Was passt, wurde assimiliert.

*APM* zeichnet sich durch eine Reihe von Merkmalen gegenüber den meisten agilen Frameworks aus. Im Wesentlichen sind das:

- *APM* integriert die aktuellen Ansätze zu einem agilen Requirements Engineering sowie zu einer Use-Case-Analyse und -Modellierung und nutzt damit bewährte und verbreitete Techniken.
- Es ist kompatibel zu verbreiteten Rollenmodellen und kann daher ohne organisatorische Probleme schnell aufgesetzt werden. Es bietet damit

auch für die Mitarbeiter fachliche Karrieremöglichkeiten, die bei anderen agilen Vorgehensmodellen oft vermisst wird.

- Softwarearchitektur ist in ihrer Umsetzung und in die Entwicklungsprozesse vollständig integriert, sodass sie den Stellenwert erhält, der für langfristig erfolgreiche Produkte notwendig ist.
- *APM* beinhaltet neben einem testgetriebenen Vorgehen das agile Testkonzept nach Crispin und Gregory [37], das weit über ein rein testgetriebenes Vorgehen hinausgeht und in dem die Entwicklung begleitende Tests von Anfang an für eine hohe Qualität sorgen.
- Wege zur angemessenen Skalierung wachsender Projekte werden mitgeliefert und müssen nicht mühsam entwickelt und integriert werden.

## 1.2 Ein paar grundlegende Begriffe vorab

Beim Erklären und Darstellen agilen Vorgehens stoßen wir, wie wir bereits im vorherigen Abschnitt gesehen haben, schnell auf ein rekursives Problem. Wir brauchen einerseits definierte Begriffe, um das Zusammenspiel in agilen Projekten zu erklären, und andererseits das Zusammenspiel in agilen Projekten, um diese Begriffe zu definieren.

Wir versuchen auch für dieses Problem eine iterative Lösung zu finden. Daher beginnen wir mit kurzen Definitionen einiger Begriffe, um danach gleich in die Grundlagen von Agilität vorzudringen.

Eine **Iteration** ist ein vorab festgelegter, fester Zeitraum, in dem wir ein Zwischenziel erreichen wollen. Iterationen dienen uns dabei, ein Projektergebnis schrittweise zu erreichen und diese Schritte zeitlich zu gliedern. Den Teil des Projektergebnisses, der im Laufe einer Iteration neu hinzukommt, nennen wir **Inkrement**. Typische Längen einer Iteration liegen in *APM* zwischen zwei und vier Wochen. In der Regel gibt es in einem Projekt mehrere Iterationen, die direkt aneinander anschließen.

Die Anforderungen an ein Projektergebnis halten wir in einer Liste fest, dem **Backlog**. Wörtlich übersetzt bedeutet Backlog »Auftragsbestand« oder »Arbeitsrückstand«. Das Besondere an einem agilen Auftragsbestand ist seine Ordnung. Die einzelnen Aufgaben, die im Backlog stehen bzw. dort referenziert sind, haben wir eindeutig geordnet. Das wichtigste Ordnungskriterium ist dabei die Bedeutung des Eintrags für den Wert des späteren Projektergebnisses aus Kunden- bzw. Anwendersicht. Des Weiteren ist jeder Eintrag im Backlog geschätzt. Ein Backlog ist also eine geordnete Liste von geschätzten Aufgaben bzw. Anforderungen für ein Projektergebnis.

Die einzelnen geordneten Einträge im Backlog beschreiben die Anforderungen an das Projektergebnis bzw. seine fachlichen oder technischen Pro-

duktmerkmale. Eine weitverbreitete und praktikable Form der Beschreibung eines Eintrags im Backlog ist die sogenannte **User Story**. Um eine solche Anforderung umzusetzen, ist eine Reihe von Tätigkeiten durchzuführen, die **Tasks** genannt werden. Dieser Unterschied zwischen User Stories im Backlog und Tasks ist deshalb wichtig, weil wir formale Kriterien an einen Backlog-Eintrag stellen, die nicht für die Tasks gelten. Wir benötigen beide Sichten, Backlog-Eintrag und Task, um ein agiles Projekt steuern zu können oder das Projektcontrolling durchzuführen.

Einen Zwischenstand unserer kompilierten und zusammengebauten Software wird **Build** genannt. Mindestens einmal pro Tag bzw. Nacht erstellen wir einen Build auf Basis der aktuellen Versionsstände in unserer Konfigurationsverwaltung (Daily bzw. Nightly Build). Noch vorteilhafter sind kontinuierliche Build-Managementsysteme wie z. B. Jenkins. Lässt sich ein Build nicht erstellen, liegt ein sogenannter Build Breaker vor. Auf einem der letzten Builds innerhalb einer Iteration beruht dann das Inkrement. Dieser Build dient damit als Besprechungsgrundlage im Ergebnisreview. Ausgewählte Builds werden in einem **Release** freigegeben und an den Kunden als **Lieferung** (engl.: Delivery) ausgeliefert.

Am Ende einer Iteration finden zwei Meetings statt, das Ergebnisreview und die Retrospektive. Im **Ergebnisreview** demonstrieren wir Mitarbeitern der Auftraggeber- bzw. Kundenseite und anderen Stakeholdern den Fortschritt aus der gerade ablaufenden Iteration. Diese Gelegenheit für Rückmeldungen zum aktuellen Projektergebnis sind essenziell für die konkrete Planung der nächsten Iteration und darüber hinaus für die Möglichkeit, das Projektergebnis für den Kunden bezüglich Kosten, Termin oder anderer Kriterien maßzuschneidern.

Die Durchführung einer **Retrospektive** ist die letzte Aktion in einer Iteration. Wir nutzen sie, um auf den Projektverlauf und die Zusammenarbeit zurückzublicken, oft auftretende Muster zu erkennen, daraus Erkenntnisse abzuleiten und kurzfristig umsetzbare Maßnahmen zu initiieren, um den Projektverlauf zur nächsten Iteration weiter zu verbessern. Während im Review das außen sichtbare Projektergebnis im Zentrum der Betrachtung steht, liegt der Fokus der Retrospektive auf den inneren Abläufen im Projekt.

Zur Steuerung kommen noch zwei verwandte, aber doch in Wirkung und Einsatzbereich unterschiedliche Konzepte hinzu: Timebox und Meilenstein. Beide verknüpfen einen gepanteten Inhalt mit einer Dauer bzw. einem Endtermin. Bei einer **Timebox** ist dabei der Termin fest. So definieren wir feste Prüfpunkte, an denen wir den erreichten Inhalt bewerten und ggf. für fehlende Teile den Restaufwand schätzen und deren Bearbeitung einplanen. Eine Iteration ist ein Beispiel für eine Timebox.

Ein **Meilenstein** definiert das Ende einer Phase oder eines Schrittes, zu dem ein bestimmter inhaltlicher Umfang erreicht sein soll. Ist der

gewünschte Umfang nicht erreicht, wird der Restaufwand geschätzt und der Meilensteintermin entsprechend verschoben.

Das wird erst einmal für die Begriffserläuterung genügen. Einen ersten Überblick über das zeitliche Zusammenspiel der Begriffe sowie der dahinter stehenden Konzepte finden Sie in Abbildung 1-2. Die Vorgänger- und Folgeiterationen sind angedeutet und grau eingezeichnet. Als Vorgriff finden Sie eine Verteilung der Aufgaben bzw. Verantwortlichkeiten auf Rollen in APM, die links in Abbildung 1-2 dargestellt sind.

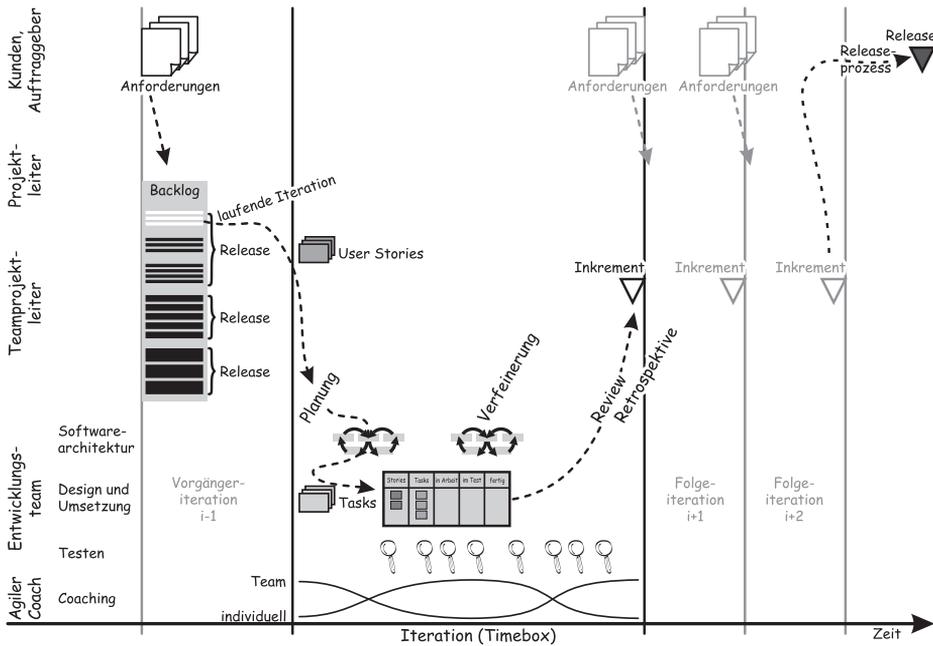


Abbildung 1-2: Das inkrementell-iterative Vorgehen in APM

Die **Planung** einer Iteration (engl. Planning) erfolgt in einer Timebox von wenigen Stunden. Sie kann und soll damit auch nicht vollständig sein. Um eine ausreichende Flexibilität zu erhalten, auch innerhalb einer Iteration auf unvorhergesehene Aspekte angemessen reagieren zu können, erfolgt in der zweiten Hälfte der Iteration meist eine weitere **Verfeinerung** der Planung (engl. Refinement) (Abb. 1-2).

## 2 Was ist Agilität?

### 2.1 Etwas Systemtheorie zur Einstimmung

Warum ist Agilität überhaupt ein Thema für das Projektmanagement? Weil wir es mit komplexen Projekten zu tun haben! Agilität ist die Antwort auf die Frage nach der Steuerung komplexer Systeme. Um zu beantworten, warum Agilität so wertvoll ist, stellen wir drei Fragen:

1. Was zeichnet Komplexität und komplexe Systeme aus?
2. Wieso ähneln Softwareprojekte komplexen Systemen?
3. Wie können komplexe Systeme gesteuert werden?

Wir werden dabei sehen, dass agile Vorgehensweisen wie Scrum oder *APM* auf den systemtheoretischen Prinzipien zur Steuerung komplexer Systeme beruhen. Nicht ausschließlich, aber doch fundamental. Daher tragen die folgenden Betrachtungen zu einem tieferen Verständnis von Agilität bei.

#### 2.1.1 Komplexe Systeme und retrospektive Kohärenz

Einen Regelungskreis wie er z. B. bei einer modernen Heizungssteuerung mit Außentemperatursensor implementiert ist, bezeichnen wir als ein kompliziertes System. Es verhält sich innerhalb der spezifizierten Parameter vorhersagbar. Ein komplexes System dagegen verhält sich nicht mehr vorhersagbar. Beispiele für komplexe Systeme sind das menschliche Verhalten Einzelner oder in einer Gruppe oder ein größeres Softwaresystem. Damit ist auch ein Softwareprojekt ein komplexes und nicht vorhersagbares System. Die Theorie komplexer Systeme ist daher von zentraler Bedeutung für die Durchführung von Softwareprojekten. Glücklicherweise brauchen wir nicht zu tief in die Theorie einzusteigen, da ein agiles Framework wie *APM* die entsprechenden Grundregeln bereits umsetzt. Zum Verständnis von *APM* trägt es jedoch bei, einige dieser Regeln und Prinzipien zu kennen.

Komplexe Systeme basieren auf Kausalketten und sind damit nicht chaotisch. Die Kausalketten sind nur derartig komplex miteinander verbunden, dass sich Ursache und Wirkung nicht vorhersagen lassen. Jede Ursache ist zugleich Wirkung und jede Wirkung wiederum Ursache. Die

für eine sichtbare Wirkung relevanten Ursachen lassen sich für uns nur im Nachhinein erkennen. Solche Erkenntnisgewinne können nur rückblickend gewonnen werden. Man kann daher sagen, dass das Verhalten eines komplexen Systems erst im Rückblick kohärent<sup>1</sup> wird. Diesen Weg des Erkenntnisgewinns bezeichnet man daher als *retrospektive Kohärenz* [170].

Wie ist nun ein komplexes System definiert? Es verhält sich nicht vorhersagbar, basiert auf Selbstorganisation, Prozesse laufen irreversibel ab und es zeigt emergentes Verhalten. Um diese vier Eigenschaften zu beschreiben, betrachten wir ein Softwaresystem und sein Umfeld.

Obwohl wir nach den neuesten Erkenntnissen über Softwarearchitektur und -design auf der Basis von Konzepten und Modellen Software entwickeln, treffen die Eigenschaften komplexer Systeme auf moderne Softwaresysteme zu. So ist z. B. das Laufzeitverhalten **nicht vorhersagbar**. Wir müssen die Performanz unter definierten Bedingungen messen. Und wenn wir das getan haben, wissen wir immer noch nicht, wie sie sich auf einem anderen Server oder mit deutlich mehr parallelen Nutzern verhält.

Die einzelnen Teile wirken nicht linear aufeinander. So ist bei einer Messung von Antwortzeiten das Verhalten nicht linear von der Anzahl der Nutzer abhängig, sondern lässt sich meist stufenförmig beschreiben und springt schlagartig auf deutlich höhere Antwortzeiten, sobald bestimmte Schwellwerte an parallelen Benutzern erreicht sind. Auf anderer Hardware oder in einem anderen Netzwerk verändern sich diese Schwellwerte sofort. Die Software verhält sich nicht vorhersehbar. Ein anderes Beispiel sind die sogenannten Seiteneffekte, also Fehler, die sich an völlig anderen Stellen in der Software ergeben als denen, an denen wir gerade gearbeitet haben.

Die **Selbstorganisation** ist ein Aspekt, der noch selten in der Softwareentwicklung zu finden ist. Dynamische Lastverteilungslösungen sind ein Beispiel für solche Ansätze. Ebenso fallen z. B. die vielen herstellerepezifischen Lösungen zur Verteilung der Schreibzugriffe bei SSDs (Solid State Disks) in diese Kategorie. Damit wird die Lebensdauer dieser Massenspeicher verlängert, was völlig transparent für das Betriebssystem oder Anwendersoftware abläuft und je nach Art und Weise der konkreten Nutzung zu unterschiedlichen Datenverteilungen führt. Die SSD hat also eine interne Komplexität und wirkt wechselseitig mit Hard- und Software, die eine Umweltkomplexität schaffen. Weitere Beispiele sind das Event-Verhalten vieler Programmiersprachen, das nicht vorhersagbare Verhalten bei Nebenläufigkeit von Java oder die internen Abläufe des Garbage Collectors sowie die nicht vorhersagbare Reihenfolge in der Abarbeitung von Aspekten in der aspektorientierten Programmierung (AOP).

Da ist die **Irreversibilität** schon weiter verbreitet. Sonst würden wir z. B. durch das probeweise Installieren und nachfolgende Deinstallieren von

---

<sup>1</sup>lat.: cohaerere – zusammenhängen.

Softwareprogrammen nicht nach und nach die Leistung eines PC ruinieren. Zugegeben, der Grad dieses Effekts hängt vom Betriebssystem ab, doch ab und an sollte man Rechner neu aufsetzen, wenn man sie über lange Zeiträume leistungsfähig nutzen möchte. Manche Entwickler haben sich bereits mit Problemen der Irreversibilität auseinandergesetzt, wenn sie Undo-Funktionalität implementiert haben. Oft kann ein echtes Undo nur durch die Rückführung auf gespeicherte Systemzustände angenähert werden.

**Emergenz** ist ein Aspekt für Software, der erst in letzter Zeit mehr und mehr Beachtung findet. Man bezeichnet mit diesem Begriff das spontane Herausbilden neuer Eigenschaften eines Systems, die sich durch das Zusammenwirken seiner einzelnen Elemente ergeben [180]. Umgangssprachlich ausgedrückt: »Das Ganze ist mehr als die Summe seiner Teile.« Emergenz kann z. B. die Architektur des Systems betreffen, wenn dieses inkrementell nach einem agilen Prozess entwickelt wird. Bestimmte Architektur-aspekte ergeben sich erst aus dem Entwicklungsprozess und dem Zusammenspiel der Komponenten. Daraus entstehen interessante Möglichkeiten für die innere Qualität der Software oder auch für ihre Flexibilität, die kaum planbar sind.

Alles in allem ähnelt moderne Software in zunehmendem Maß sehr stark komplexen Systemen. Diese Eigenschaften unserer Projektergebnisse wirken sich auch auf das Projektmanagement aus. Dazu kommt, dass sich die Entwickler sowie die aus ihnen gebildeten Entwicklungsteams exakt so verhalten wie ein komplexes System. Auch hier gelten dieselben Regeln. Selbstorganisation findet stets zu einem gewissen Grad in der Interaktion innerhalb von Gruppen statt. In agilen Projekten wird diese explizit gefordert und gefördert. Jeder Mensch im Team lernt permanent und beurteilt z. B. Situationen zu späteren Zeitpunkten oft anders als zu früheren. Mit der neuen Beurteilung ändert sich meist auch die Reaktion bzw. Handlung.

Wir erkennen an diesen Beispielen auch, dass wir zwischen einer internen Komplexität und der Komplexität der Umwelt unterscheiden können. Betrachten wir also ein Softwareprojekt, so haben wir durch das Projektmanagement und die Entwicklungsteams sowie das schrittweise wachsende Softwareprodukt eine interne Komplexität. Über unsere Auftraggeber, spätere Anwender, Kunden oder andere Projekte entsteht eine Umweltkomplexität. Beide Aspekte verhalten sich nicht vorhersagbar.

### 2.1.2 Die Steuerung komplexer Systeme

Wenn wir das Verhalten eines komplexen Systems nicht vorhersagen können, lässt es sich auch nicht mit traditionellen Konzepten steuern. Sich hinzusetzen, einen Plan zu schmieden und den dann strikt einzuhalten, verspricht kaum Erfolg. Für die Optimierung von Abläufen oder andere Veränderungen in komplexen Systemen, wie die Durchführung eines Soft-

wareprojekts, ist der rückblickende Erkenntnisgewinn, die retrospektive Kohärenz, die einzige Möglichkeit, Erklärungen und Zusammenhänge zu erkennen. Das hat Konsequenzen für die Strukturierung eines Projekts, die einen Großteil der *APM*-Methodik ausmacht.

Zur Steuerung benötigen wir in kurzen Abständen rückblickende Analysen auf unterschiedlichen Ebenen. Dabei unterscheiden wir zwischen direkten Rückkopplungen, wie sie z. B. über die Unit Tests in der testgetriebenen Entwicklung erfolgen, und den länger laufenden, kalibrierenden Rückkopplungsschleifen, mit denen wir uns z. B. über die Ergebnisreviews oder Retrospektiven am Ende jeder Iteration immer wieder aktuell auf den Projekterfolg ausrichten. Diese Rückkopplungen sind essenziell zur Steuerung von komplexen Projekten und damit unverzichtbar. Auf keinen Fall dürfen sie aus Bequemlichkeit oder vermeintlichen Zeitgründen gekürzt werden oder gar ganz entfallen. Über die kalibrierenden Rückkopplungen vermindern wir ein häufiges Risiko in agilen Projekten, den kurzfristigen Aktionismus. Würden wir ein Projekt nur über kurze, direkte Rückkopplungen steuern, verlören wir leicht das Ziel aus den Augen und wären vorwiegend ereignisgetrieben. Über die länger laufenden Rückkopplungen erfolgen Lernschleifen, die für das Projektmanagement ebenso wichtig sind wie für die Entwicklungsteams.

### 2.1.3 Selbstorganisation und Führung

Selbstorganisation ist ein Merkmal komplexer Systeme. Sie kann uns helfen, z. B. mit der Umweltkomplexität angemessener zurechtzukommen. Im Vordergrund eines selbstorganisierten Teams stehen die Prozesse, die im Team ablaufen. Die innere Ordnung entwickelt sich dabei permanent weiter. Alle Strukturen haben daher nur temporären Charakter. Bis zu einem gewissen Grad ist Selbstorganisation dabei eine Eigenschaft jeder Gruppe. Der Grad und die Konsequenz ihrer Umsetzung variieren jedoch stark [75]. Typischerweise sprechen wir im agilen Projektmanagement erst ab einem deutlich sichtbaren Grad von selbstorganisierten Teams.

Solche Teams sind z. B. cross-funktional besetzt und bestehen aus vielen Teammitgliedern mit im günstigsten Fall jeweils mehreren Fähigkeiten, sodass die Aufgaben flexibel im Team übernommen werden (Abschnitt 4.4.3). Es erfolgt keine explizite Verteilung von Aufgaben durch eine Führungskraft oder einen Coach, sondern die Teammitglieder wählen sich ihre Tasks selbst aus (Pull- anstatt Push-Prinzip) (Abschnitt 18.3).

In selbstorganisierten Teams wird die innere Struktur nicht von außen z. B. über einen Projektleiter vorgegeben, sondern situativ von den Teammitgliedern gebildet. Daraus erwachsen die Stärke und die große Flexibilität selbstorganisierter Teams. Ihre Realisierung stellt jedoch hohe Ansprüche an die einzelnen Teammitglieder. Ein essenzieller Aspekt ist dabei

die Orientierung anhand der Produktvision und der Projektziele, über den die Ausrichtung des selbstorganisierten Teams erfolgt.

Selbstorganisation stellt derzeit die wohl beste Lösung zur Umsetzung dynamischer und komplexer Projekte dar. Über die hohe Flexibilität und Wirk-, Handlungs- und Kommunikationsmöglichkeiten eines selbstorganisierten Projektteams können wir bestmöglich auf die Anforderungen aus dem komplexen Umfeld des Projekts reagieren.<sup>2</sup>

Was hat das mit agilem Projektmanagement und *APM* zu tun? Eine mit Spezialisten aufgestellte, hierarchische Projektorganisation hat eine zu geringe Flexibilität bezogen auf die Vielfalt der Anforderungen aus der Umwelt des Projekts. Die Einflüsse auf das System *Projekt* treiben eine Menge grundlegender Variablen wie Funktionalität, Termin, Qualität und Kosten aus dem gewünschten Rahmen, wenn aus dem System *Projekt* heraus nichts dagegen getan wird.

Die Lösung besteht darin, über das Mittel der Selbstorganisation die geballte Kraft des Wissens aller Teammitglieder zu nutzen, um angemessene Lösungen zu entwickeln. Damit dies funktioniert, liefert die Führung die Zusammenhänge und Rahmenbedingungen als grundlegende Orientierung. Die notwendige Flexibilität im Projektalltag, mit Störungen umzugehen, wird durch eine vergleichsweise große Anzahl an Universalisten im selbstorganisierten Projektteam durch das Team selbst erreicht.

In agilen Projekten findet also sehr wohl *Führung* statt. Sie verhält sich jedoch anders als in hierarchisch organisierten Strukturen. Die Führung in einem agilen Projekt gibt das Ziel vor, verfeinert es regelmäßig und definiert den Rahmen, in dem das Ziel erreicht werden soll. Dazu gehören die grundsätzliche Vorgehensweise sowie Zeit- und Budgetvorgaben. Der konkrete Weg zur Umsetzung des Projekts wird von den Teammitgliedern im Rahmen der inkrementell-iterativen Entwicklung gefunden. Daher sind die Lernschleifen am Ende jeder Iteration von so großer Bedeutung. Eins können wir jedoch sicher ausschließen: Einfach nur steuern werden wir ein Softwareprojekt nicht können. Sonst wäre es per definitionem kein Projekt.

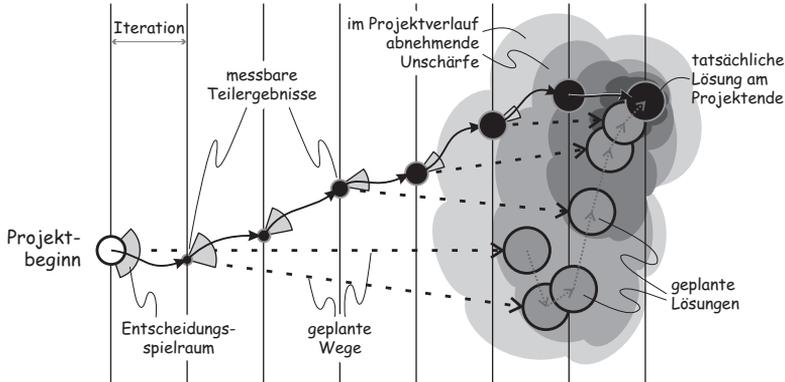
#### 2.1.4 Der Mythos von den sich ändernden Anforderungen

Als Begründung für den Wunsch nach Agilität in der Softwareentwicklung höre ich oft, dass damit besser auf die sich so oft ändernden Anforderungen reagiert werden soll. Ja, man hört und liest dieses Argument so oft, dass wir geneigt sind, es einfach hinzunehmen und nicht weiter zu hinterfragen. Es ist jedoch für ein agiles Projektmanagement wichtig, auch hier genauer hinzusehen. Ändern sich Anforderungen wirklich so oft bzw. was ändert sich bei

---

<sup>2</sup>Wir können diesen Zusammenhang als eine praktische Anwendung des Ashby'schen Gesetzes aus der Kybernetik betrachten [50].

den Anforderungen genau? Natürlich können sich im Laufe der Zeit Anforderungen inhaltlich oder vom Umfang her ändern. Viel häufiger treffen wir jedoch auf einen anderen Zusammenhang: Die Anforderungen werden im Laufe der Zeit immer detaillierter und konkreter. Nicht der Inhalt ändert sich, sondern das Abstraktionsniveau (Abb. 2-1) [118].



**Abbildung 2-1:** Schrittweise Annäherung und die Wolkenmetapher: Die Unschärfe in den Anforderungen nimmt mit jeder Iteration ab [118].

Wie wir bereits gesehen haben, ist auch die Projektumwelt ein komplexes System. Damit stehen auch die Fachexperten oder Produktmanager vor dem Problem, die Komplexität ihrer Anforderungen zu erfassen und zu beherrschen. Und dabei gehen sie automatisch vom Abstrakten zum Konkreten. Nur das, was im ersten Fokus eines neuen Produkts liegt, kann bereits tiefer erfasst werden. Bei vielen anderen Aspekten bleiben sie zwangsläufig noch an der Oberfläche. Oft müssen die Stakeholder des Projekts erst Teile der laufenden Software sehen und ausprobieren, um die nächsten Entscheidungen treffen und weitere Konkretisierungen vornehmen zu können. Genau diese *Moving-Target*-Dynamik berücksichtigen wir in einem agilen Projekt unter anderem mit dem inkrementell-iterativen Vorgehen (Abb. 2-1).

Neben dem Schema des iterativen Vorgehens ist in der Abbildung auch eine andere Konsequenz für die Softwareentwicklung angedeutet. Mit jeder Iteration schränken wir durch die in der Iteration getroffenen Entscheidungen den Entscheidungsspielraum für die folgenden Iterationen etwas ein. Dies ist ganz normal, doch sollten wir versuchen, den Verlust an Spielraum zu verlangsamen bzw. so weit wie möglich zeitlich nach hinten zu schieben. Insbesondere für Architekturentscheidungen sowie grundsätzliche Rahmenbedingungen stellt sich dieser Zusammenhang besonders einschränkend dar. Wir kommen in Abschnitt 12.3.1 darauf zurück und überlegen dort, wie wir damit umgehen können.

Zu guter Letzt erkennen wir in Abbildung 2-1 auch, warum ein starres Festhalten an dem ersten Plan, der sogenannten Baseline, so fatale Konsequenzen haben kann. Das Ziel bewegt sich und die Baseline kann nicht mehr zum Erfolg führen. Eine komplette Vorabanalyse hilft uns aber auch nicht weiter, da die Ansprechpartner ihre Anforderungen noch nicht einheitlich tief konkretisieren können. Ihnen fehlt die Information aus den Inkrementen, um ihre weiteren Entscheidungen treffen zu können. Die Baseline ist wichtig, um erste Abschätzungen machen zu können, um z. B. Wirtschaftlichkeitsbetrachtungen von möglichen Projekten durchführen zu können. Als langlebiger Plan taugt sie jedoch kaum, denn ein komplexes Projekt lässt sich nicht genau genug vorhersagen. Als Konsequenz aus dieser Erkenntnis planen wir fortlaufend bzw. iterativ und passen unseren Plan, das Ziel zu erreichen und die Produktvision umzusetzen, stets an die aktuellen Erkenntnisse an.

## 2.2 Agile Werte, Prinzipien und Praktiken

Wieso können wir mit agilen Ansätzen angemessener mit der Komplexität von Projekten und den Teams umgehen als auf traditionellen Wegen? Agile Rahmenwerke basieren auf einer grundsätzlich anderen Grundannahme zur Durchführung von Projekten. Die Erkenntnisse aus der Systemtheorie zur Unvorhersehbarkeit komplexer Zusammenhänge als Rahmenbedingungen werden akzeptiert und es wird nicht versucht, durch immer tieferes hierarchisches Gliedern und eine Mikroplanung die Probleme und Risiken von Softwareprojekten vorwegzunehmen.

Die Selbstorganisation der einzelnen Teams und ihrer Mitglieder sind das stärkste Werkzeug im Umgang mit unvorhersehbaren Aspekten im Projektablauf und das beste Mittel, die Flexibilität der Projektorganisation zu optimieren. Wir versuchen, jeden beteiligten Menschen in die Lösung einzubinden und seine spezifische Erfahrung dabei zu nutzen. Kunden, Anwender und andere Stakeholder mögen vielleicht Teil des Problems sein, sie sind aber auf jeden Fall Teil der Lösung! Insofern ist ein agiles Projektmanagement wie *APM* die Umsetzung der systemtheoretischen Erkenntnisse in die Realität von komplexen Projekten.

### 2.2.1 Werte, Prinzipien und Praktiken im Zusammenspiel

Im Folgenden werden Werte und Prinzipien diskutiert und auch bereits einige agile Praktiken wie testgetriebenes Vorgehen oder Continuous Integration erwähnt. Wie hängt das alles zusammen?

Werte sind etwas Übergreifendes, was ein einzelner Mensch als wichtig und lohnend einzuschätzen lernt. Ein Wert kann ein Lebensprinzip sein

oder etwas, was man erreichen bzw. erhalten möchte [189]. Unsere Werte bilden damit eine Art oberste Leitlinie für unsere Entscheidungen und unser Handeln. Wir können aus unseren Werten Prinzipien ableiten, also Zusammenhänge, die für uns, in einem Kontext interpretiert, konkreter als die zugrunde liegenden Werte eine Entscheidungsbasis bilden. Prinzipien manifestieren damit unsere Werte.

Praktiken sind konkrete Techniken oder Methoden, mit denen wir Prinzipien umsetzen. Es gibt damit zahlreiche Praktiken, die wir differenziert einsetzen können, und es kommen regelmäßig neue hinzu. Wir können damit allerdings nicht aus dem Einsatz einer oder mehrerer agiler Praktiken darauf schließen, dass wirklich agil vorgegangen wird. Dazu betrachten wir, inwiefern durch die agilen Praktiken die agilen Prinzipien im Sinn der agilen Werte wirklich erreicht bzw. umgesetzt werden. Dieser Zusammenhang macht auch eine objektive Bewertung agiler Vorgehensweisen so schwierig. Anders als bei einem schwergewichtigen Vorgehensmodell wie z. B. dem Rational Unified Process (RUP) können nicht alle Praktiken konkret beschrieben und in einen eindeutigen Zusammenhang gebracht werden, nach dem sie anzuwenden sind. Über die agilen Praktiken ergeben sich eine Reihe flexibel im Projektkontext nutzbarer Möglichkeiten.

### 2.2.2 Agile Werte

Wie alle agilen Frameworks basiert auch *APM* auf dem Agilen Manifest für agile Softwareentwicklung [12] aus dem Jahr 2001. Dazu kommen die Werte aus dem eXtreme Programming oder kurz XP.

### eXtreme Programming – XP

Kent Beck, Ron Jeffries und Ward Cunningham, drei der 17 Erstunterzeichner des Agilen Manifests, setzen mit ihrem agilen Framework XP aus der zweiten Hälfte der 1990er-Jahre, das Ende 2004 noch einmal überarbeitet wurde, auf fünf Werten auf [11]:

- Kommunikation – Der Kommunikationsfluss wird über eine Vielzahl von Verfahren von Unit Tests bis hin zu regelmäßigen Meetings aufrechterhalten.
- Einfachheit – Es wird versucht, die einfachste mögliche Lösung zu finden, die vermutlich ausreichen wird, um einen unnötigen Erststellungs- und Wartungsaufwand zu vermeiden. Einfachheit wird auch oft als KISS-Prinzip (Keep it small and simple) bezeichnet. Dahinter steckt die Erkenntnis, dass Software eher selten geschrieben, dafür aber umso häufiger gelesen wird, was betriebswirtschaftlich auch dadurch sichtbar wird, dass in ein Produkt ca. 20 % der gesamten Kosten in die initiale